Synthesizing Tabular Data using Conditional GAN

by

Lei Xu

B.E., Tsinghua University (2017)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2020

Signature redacted

Author . . . .
Department of Electrical Engineering and Computer Science
January 30, 2020

Signature redacted

Certified by . . . .
Kalyan Veeramachaneni
Principal Research Scientist
Laboratory for Information and Decision Systems
Thesis Supervisor

Signature redacted

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Synthesizing Tabular Data using Conditional GAN

by

Lei Xu

Submitted to the Department of Electrical Engineering and Computer Science
on January 30, 2020, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

In data science, the ability to model the distribution of rows in tabular data and generate realistic synthetic data enables various important applications including data compression, data disclosure, and privacy-preserving machine learning. However, because tabular data usually contains a mix of discrete and continuous columns, building such a model is a non-trivial task. Continuous columns may have multiple modes, while discrete columns are sometimes imbalanced, making modeling difficult.

To address this problem, I took two major steps.

(1) I designed `SDGym`, a thorough benchmark, to compare existing models, identify different properties of tabular data and analyze how these properties challenge different models. Our experimental results show that statistical models, such as Bayesian networks, that are constrained to a fixed family of available distributions cannot model tabular data effectively, especially when both continuous and discrete columns are included. Recently proposed deep generative models are capable of modeling more sophisticated distributions, but cannot outperform Bayesian network models in practice, because the network structure and learning procedure are not optimized for tabular data which may contain non-Gaussian continuous columns and imbalanced discrete columns.

(2) To address these problems, I designed `CTGAN`, which uses a conditional generative adversarial network to address the challenges in modeling tabular data. Because `CTGAN` uses reversible data transformations and is trained by re-sampling the data, it can address common challenges in synthetic data generation. I evaluated `CTGAN` on the benchmark and showed that it consistently and significantly outperforms existing statistical and deep learning models.

Thesis Supervisor: Kalyan Veeramachaneni
Title: Principal Research Scientist
Laboratory for Information and Decision Systems

Acknowledgments

Foremost, I would like to thank my advisor, Dr. Kalyan Veeramachaneni for the continuous support of my study and research, for his patience, motivation, enthusiasm, and immense knowledge. I would like to thank all members in the *Data to AI Lab* for inspiring discussion. During this research project, Carles Sala helped with maintaining the GitHub repo; Cara Giaimo helped with text editing; Arash Akhgari helped with figure design; I would like to thank them for their contribution and help. Finally, I would like to thank my family and friends who have been supported me all the time.

Contents

1 **Introduction** 15

1.1 Tabular data in different domains 15

1.1.1 Recommender systems 16

1.1.2 Healthcare 18

1.1.3 Data storage and disclosure 19

1.2 Necessity of synthetic data 20

1.3 Types of synthetic data generators 22

1.4 Existing generation-based methods 23

1.5 An overview of this research 24

2 **Primer on Generative Models** 27

2.1 Variational autoencoders 27

2.2 Generative adversarial networks 29

3 **Synthetic Data Generation Task and Related Work** 35

3.1 Synthetic data generation 35

3.2 Existing techniques to generate synthetic data 37

3.2.1 PrivBayes 38

3.2.2 MedGAN 40

3.2.3 TableGAN 41

3.2.4 VeeGAN 42

4 **Challenges of Modeling Tabular Data using GAN** 47

4.1 Challenges on single-table non-time-series data 47
4.2 Challenges on time-series data 51
4.3 Challenges on multiple table data 51

5 Conditional Tabular GAN 53
5.1 Notations 53
5.2 Mode-specific normalization 53
5.3 Conditional tabular GAN architecture 56

6 Other Methods for Synthetic Data Generation 63
6.1 TGAN 63
6.2 Tabular VAE 66

7 SDGym Benchmark Framework 69
7.1 Simulated data 70
7.2 Real data 71

8 Experiment Results 75
8.1 Settings and hyperparameters 75
8.2 Quantitative results 76
8.3 Case analysis 79
8.4 Ablation study 80
8.5 Discussion 81

9 Conclusion and Future Work 83

A Notations 85

B Figures 87

List of Figures

3-1 The MedGAN framework contains an encoder, a decoder, a generator, and a discriminator. The encoder and decoder are pretrained on the real data and fixed in later steps. During training, the output of the generator is passed through the decoder before feeding into the discriminator. The discriminator checks whether the data are real or fake. 41

3-2 VeeGAN framework. VeeGAN contains three modules: a generator, a discriminator, and a reconstructor. The top section shows the workflow starting from random noise z. In this workflow, the generator projects z to synthetic data x' and tries to fool the discriminator. The gradients from the discriminator help the generator improve. Meanwhile, the reconstructor learns to project x' back to $z^r = z$. The bottom section shows the workflow starting from real data x. x is inputted to the reconstructor in order to generate a representation vector z'. Then real tuples (x, z') and fake tuples (x', z) are used to train the discriminator. 44

3-3 The reconstructor makes a connection between missing modes and existing modes, so that the generator can recover from mode collapse. The left section shows how the generator projects random noise into the data space. The right section shows how the reconstructor projects data into the noise space. 45

4-1 Challenges of non-Gaussian distribution on GAN models. Assume we have a non-Gaussian continuous column with a few large outliers. The large outliers squeeze all other values towards -1. After min-max normalization, the probability density looks like (A). To use a neural network with `tanh` activation to generate this column, the probability distribution of values before the `tanh` activation looks like (B). The gradient of tanh vanishes in this range, as shown in (C). The model can not learn effectively with a vanished gradient. 48

4-2 [50] show that a vanilla GAN can not model a simple 2-dimensional Gaussian mixture. (A) is the probability density of 25 Gaussian distributions aligned as a grid. (B) is the corresponding distribution learned by GAN. (C) is the original distribution and (D) is the corresponding distribution learned by GAN. 49

5-1 An example of mode-specific normalization. The distribution of a continuous column (the blue dashed line in the left figure) has 3 modes, and these modes are modeled by a variational Gaussian mixture model. In the middle figure, a value from that column $c_{i,j}$ appears. $c_{i,j}$ has the probability density of ρ_1, ρ_2, ρ_3 of coming from each mode. It is more likely to come from the third mode. So $c_{i,j}$ is normalized by the mean and standard deviation of the third mode, namely η_3 and ϕ_3. 55

5-2 `CTGAN` structure. 57

6-1 Example of using `TGAN` generator to generate a simple table. The example has two continuous variables and two discrete variables. The order of these columns is $[C_1, D_1, C_2, D_2]$. `TGAN` generates these four variables one by one following their original order in the table. Each sample is generated in six steps. Each numerical variable is generated in two steps while each categorical variable is generated in one step. 64

6-2 TVAE structure. 68

7-1 Evaluate synthetic data generator using simulated data. 71

7-2 Real data in synthetic data generator benchmark. 72

8-1 Visualize synthesized `grid` data set using `TableGAN`, `VEEGAN`, `TVAE` and `CTGAN`. The red marks are the ground truth modes and the blue dots are synthetic samples generated by different synthesizers. 79

8-2 Visualize synthesized `gridr` data set using `TVAE` and `CTGAN`. The red marks are the ground truth modes and the blue dots are synthetic samples generated by different synthesizers. 80

B-1 Visualize `grid` (left), `gridr` (middle) and `ring` (right) datasets. Each blue point represents a row in a table. The x-axis and y-axis represent the value of two continuous columns respectively. The red 'x's in the plots are the modes for Gaussian mixtures. 87

B-2 Bayesian network structures of `asia` (left) and `alarm` (right) datasets. 88

B-3 Bayesian network structures of `child` (left) and `insurance` (right) datasets. 88

List of Tables

1.1 Data-related issues to address in different domains. 20

4.1 A summary showing whether existing methods and our CTGAN explicitly address the aforementioned challenges [C1 - C8]. (* indicates it is able to model continuous and binary.) 51

7.1 Simulated datasets in our benchmark. #C, #B, and #M mean number of continuous columns, binary columns and multi-class discrete columns respectively. 71

7.2 Real datasets in our benchmark. #C, #B, and #M mean number of continuous columns, binary columns and multi-class discrete columns respectively. 73

7.3 Classifiers and regressors selected for each real dataset and corresponding performance. 73

8.1 Benchmark results on Gaussian mixture simulated data. 77

8.2 Benchmark results on Bayesian network simulated data. 77

8.3 Benchmark results on real data. 78

8.4 Distance between synthetic data and nearest neighbor in training set. 78

8.5 Classification performance of different classifers. 79

8.6 Ablation study on mode-specific normalization. 81

8.7 Ablation study on training-by-sampling and conditional vector. 81

8.8 Ablation study on Wasserstein GAN and PacGAN. 81

A.1 Notations 86

Chapter 1

Introduction

Tabular data is one of the most common and important data modalities [1]. Massive amounts of data, including census results, health care records, and web logs (generated from human interaction with websites) are all stored in tabular format. Such data is valuable because it contains useful patterns that can help in decision-making. As more and more companies, policymakers, and research institutes rely on data to make decisions, people have recognized the need to enable good decision-making and ensure privacy protection, as well as manage other issues. It is in this context that the demand for synthetic data arises.

1.1 Tabular data in different domains

Tabular data is widely used in different fields and has become an integral part of predicting potential needs. Every day when we open YouTube, our favorite videos are already queued up and can be viewed with just one click, without the need for tedious searches. Data can be used to predict the risk of disease and provide people with life and medical advice. Data can also help governments and companies make decisions. The growth of the field is exponential, as the availability of massive data inspires people to explore different applications.

However, due to the quality, quantity, and privacy issues associated with using real data, people usually do not stick to the *original* data when creating and exploring these applications in various domains. We give examples within several representative domains to show how issues with real data motivate the need for synthetic data.

1.1.1 Recommender systems

To this day, as long as we use the Internet, we will inevitably be affected by the recommender system. A recommender system is one of the most profitable use cases for tabular data, and has enabled companies like Amazon and Google to grow to a trillion-dollar market cap. Taking movie recommendation as an example, we introduce how recommender systems are built, and the issues that arise and motivate the need for synthetic data generation methods.

Movie recommendation is a well-studied example in machine learning and data science due to the availability of datasets like MovieLens [19] and Netflix [6]. The methods verified on movie recommendation problems are also applied in other scenarios, such as product recommendations on Amazon [37, 24].

- Early content-based recommender systems [2] use movie descriptions and user profiles to make recommendations. For example, if a user is interested in a particular actor, the recommender system will recommend all the movies from that actor. To build a content-based recommender system requires a table of movie metadata, such as the directors, cast, and genres, and a table of users' watch history. In the training phase, users' watch histories are used to identify the users' interests and construct user profiles. For example, if a user watches horror movies more frequently than average, the *horror* tag will be added to the profile. In the inference phase, movies related to the profile are recommended to the user.

  Content-based methods require high-quality metadata. Before training the model, metadata are constructed manually to ensure quality. Recently, content-based methods have been replaced by collaborative filtering methods due to their superior performance.

- Collaborative filtering [47] is a common method used in recommender systems.

Collaborative filtering methods start with a user-item matrix M. Each row of M represents all the movies the user likes. An entry in a row is 1 if the user likes the movie, and 0 otherwise. Then M is factorized to a low-rank user matrix U and a low-rank movie matrix V where $M \approx UV^T$. The interest level of user i to movie j is the inner product of the user vector and movie vector $U_i V_j^T$, and recommendations are then made according to the interest level. To train a collaborative filtering model requires only a table of users' watch history.

When building models this way, the following data issues arise. **The quantity issue:** Collaborative filtering will give superior performance only when the scale of the dataset is large. To overcome small datasets, people generate large synthetic datasets. For example, to generate synthetic data from MovieLens [19], people expand each real user into several similar fake users.[1] **The quality issue:** Expanding the dataset is not necessary for an industry-scale system, because the quantity of training data is usually not an issue for companies. Instead, their main issue is data quality. Users' watch history is noisy because watching a movie does not necessarily indicate that the user likes that movie. To tackle this, [35] filters the training data using users' ratings. The movies that are rated low are taken out. Filtering is also done on users. Users with very few watched movies tend to provide a noisy signal in the training, so [21] filters out users with less than 20 watched movies. **The imbalance issue:** Furthermore, the data distribution is highly imbalanced. For example, because there are far fewer Chinese movies than English movies on most platforms, their models perform worse when it comes to recommending those movies. This holds true across minorities. [38, 34] split the dataset into smaller datasets and train local models to address these issues. **The privacy issue:** It is also important to note that in large-capacity models[2], the model can remember a lot of information, including users' personal information. This presents the risk of

[1] The code to generate synthetic MovieLens data from real data is available online. `https://github.com/mlperf/training/tree/master/data_generation`

[2] Collaborative filtering methods have lots of parameters. A typical setup uses 50 to 500 dimensional vectors to represent users and movies.

leaking sensitive user information. For example, Netflix's publicly available recommendation data set can be used to identify individuals and determine sexual orientation, and has been prosecuted [49].

- [22, 52] uses gated-recurrent networks and transformers to make sequential recommendations. In these models, the prediction task is to predict what someone will watch next, using their watch history as a sequence. These models perform better than collaborative filtering because they consider sequential information. For example, when a user watches Avengers I and II successively, he is likely to watch Avengers III next. A sequential recommender can capture this pattern, while collaborative filtering can not. To train a sequential recommender system requires a table of users' watch history with timestamps.

  The privacy issue: The parameters in these models are even larger than those in collaborative filtering models. These models are also highly non-linear and can remember various patterns, which increases the risk of a privacy breach.

Using tabular data to train a recommender system involves data quality, data imbalance, and data privacy issues. Data quality can be improved by applying various filtering criteria to the data. Data imbalance can be addressed by partitioning the data and learning of local models. Addressing the privacy issue is more challenging.

1.1.2 Healthcare

Big data is also used in the medical field to make diagnoses more accurate and efficient.

- Data can be used to predict whether a patient will show up for an appointment [3], to help the hospital improve time management and resource use.

- Data can also be used to determine whether a patient has a certain disease, such as heart disease, pneumonia, etc. This is a critical application. Due to issues with medical data, the robustness and fairness of these models are questioned.

  Collecting medical data requires the involvement of specialized doctors, leading to a high collection cost and data. [16] uses data augmentation to increase the size of a medical dataset. Medical data may also be noisy, and doctors' wrong judgments can introduce errors into the data. In addition, it suffers from data imbalance — [12] shows that the imbalanced medical data leads to unfair predictions. Medical data also contains a large amount of personal information, which can easily identify individuals and infringe on people's privacy.

- Wearables also collect large amounts of data and can provide suggestions meant to improve people's health. For example, [46] uses wearables to track people's sleeping conditions. Data collected by wearables, which can include location, sound, or even images, are extremely sensitive and should not leave the device. People design algorithms [8, 28] to train machine learning models on this data without sending it to a central server.

Machine learning systems that use healthcare data deal with very sensitive information and make critical decisions. The quantity, quality, imbalance, and privacy of this data should be handled seriously.

1.1.3 Data storage and disclosure

Sometimes, tabular data is simply constructed by a website or gathered by surveyors and is then either stored in a database, used to complete website business, or released by a statistics bureau. This data may not require the use of machine learning for analysis in the short term, but privacy issues will still be encountered during the process of construction, storage, and distribution.

- Every few years, the National Bureau of Statistics conducts a census, and the results of the census are published online. This data is helpful for solving social problems, but the direct publication of accurate census data is likely to violate citizens' privacy. If only statistical data is published instead of data for each individual, the value of the data for research will decrease. As a result, people have invested a lot of time in resolving the issue of census data while protecting privacy. Some researchers in the statistical science community have begun using randomization-based methods to perturb sensitive columns in tabular data [44, 42, 43, 32].

- Another scenario involves data storage and use within a company. When users interact with a company website, a large number of records are generated in that company's database. Users may also fill in personal information such as addresses, credit card numbers, preferences, etc. This information must be strictly confidential in accordance with the General Data Protection Regulation (GDPR). However, inside the company, engineers will have software development and bug fix requirements that require data use. Companies do not want their employees to have access to real user data, as this would violate user privacy, so actions need to be taken to prevent insiders from reading sensitive information. At Uber, for example, a system is deployed so that employees can only access perturbed customer data [29]. Prior to deploying the system, Uber reportedly mishandled customer data, allowing employees to gain unauthorized access [20].

1.2 Necessity of synthetic data

Table 1.1: Data-related issues to address in different domains.

	Quantity	Quality	Imbalance	Privacy
Recommender System	x	✓	✓	✓
Healthcare	✓	✓	✓	✓
Data Disclosure	x	x	x	✓

Despite the huge investment institutions put into collecting tabular data every day, real tabular data cannot always fulfill what is asked of it. Table 1.1 summarizes data-related issues that arise in different domains.

- **The quantity issue:** In certain areas, insufficient data is an issue. Especially if the acquisition of data requires a person with specialized skills, such as in the medical field, the amount of data and the cost of acquiring the data will become a problem. If real data can be supplemented and enhanced using synthetic data, then less existing data can be used to achieve more valuable applications.

- **The quality issue:** Data quality issues are common. During the data collection process, various factors can affect the quality of the data — for example, missing values and outliers from people incorrectly filling in a questionnaire. Learning the distribution of the data and repairing and enhancing the data can reduce the impacts of this problem.

- **The imbalance issue:** Data imbalance is the normal state of tabular data, as tables usually have major and minor categories. Imbalance causes a lot of problems when developing models. Using synthetic data to supplementing the niche data in a table can solve this problem.

- **The privacy issue:** Furthermore, most tabular data contains sensitive information that could be used to identify individuals and intrude on their privacy. Data containing sensitive information tends to be strictly protected and out of reach of researchers. If synthetic data can preserve correlations in a table but remove all sensitive information, it could be used to remove this barrier to data disclosure.

Applications of Synthetic Data High-quality synthetic data has important applications:

- **Data understanding**: Learning the distribution of tabular data can help us understand the underlying structure and association between columns.

- **Data compression**: Synthetic data generators can be used to store tabular data efficiently and compactly. A small generative neural network can be easily stored on portable devices to generate an infinite number of rows.

- **Data augmentation**: A generative model can generate (more) training data, or reasonably perturb the original data, which can improve the performance of downstream predictive models.

- **System testing**: Synthetic datasets that are derived from the same *actual underlying process* as their corresponding real datasets can be used to test new systems, as opposed to those generated from (usually) unrealistic simulating scenarios. System testing is sometimes conducted using synthetic data in order to protect privacy or prevent over-fitting. The use of high-quality synthetic data can ensure similar performance in testing and production environments.

- **Data disclosure**: Data privacy is an important issue today. Using synthetic data instead of real data can avoid the disclosure of private information while still allowing data to be used for various applications.

1.3 Types of synthetic data generators

Synthetic data generation is an effective way to address these quantity, quality and privacy issues. We categorize synthetic data generation methods into two stages:

- **Perturbion-based methods.** Methods in this category modify the values in existing tables to fix outliers or reduce privacy leaks. These methods have been studied for many years. Because each row in synthetic data generated with this method has a corresponding row in the real data, these methods can neither increase the size of the data nor provide good privacy protection.

- **Generation-based methods.** Another category of methods tries to generate synthetic data from some distribution. This distribution could either be handcrafted or learned from data. These methods can generate an arbitrary amount of data. Under certain circumstances, privacy-protecting mechanisms can be added to provide better privacy. Generating data with handcrafted distributions is in wide use, while synthesizing data using learned distributions is an area of recent study.

The methods in the first stage are an ad-hoc, ineffective solution to protect privacy, while methods in the second stage can systematically address quantity, quality and privacy issues by substituting real data with synthetic data.

1.4 Existing generation-based methods

Huge efforts have been made in the field of synthesizing tabular data from existing distributions. Both statistical methods and deep learning methods are used to learn the distribution of real data so that synthetic data can be generated by sampling.

Statistical methods use a family of the predefined probability distributions to fit a new tabular dataset. For example, a Gaussian mixture model can be used to model the joint distribution of a few continuous columns, while Bayesian networks can be used to model the joint distribution of discrete columns. However, because these models are limited by the probability distributions available, they are not general enough for various datasets, especially when the data has a mix of discrete and continuous columns. In the case of modeling survey data, where continuous columns can hardly be avoided, people discretize continuous columns so that the data can be modeled by Bayesian networks or decision trees. Even beyond issues with model capability, training such statistical models is expensive, and these models cannot scale up to large-scale datasets with thousands of columns and millions of rows.

Deep learning methods make up another major category of data synthesizers. The motivation for building a high-quality deep neural network for tabular data comes from the success of such models on computer vision and natural language processing. Deep generative models like variational autoencoders (VAEs) [31] and generative adversarial networks (GANs) [17] have two new capabilities: First, the capacity to learn a complicated high-dimensional probability distribution, and second, the ability to draw high-quality samples from images or text. These capabilities have enabled various important applications in vision and language [26, 59, 33, 57, 54]. It is also possible to build similarly high-quality models to generate synthetic tabular data — the implicit joint distribution of columns can be learned from real data, and synthetic rows can then be sampled from that distribution. A few models have been proposed (`MedGAN` [13], `TableGAN` [39], `PATE-GAN` [30]) that work by directly applying fully connected networks or convolutional neural networks on tabular data without considering the specific case of modeling tabular data. These models can perform well on datasets, but have not been thoroughly compared with existing statistical models.

1.5 An overview of this research

In this project, we focus on using deep generative models to model the probability distribution of rows in a table. To thoroughly understand the complexity of the problem, we designed a benchmark framework with simulated datasets and real datasets. (See chapter 7.) This framework, called SDGym, can automatically evaluate all models over all the datasets, and can help understand the properties of different tabular data and their effects on different models. We also implemented several existing methods, and evaluate all methods over different datasets and metrics.

By comparing existing models, we found that despite the potential of GANs to model arbitrary distribution, GAN-based models cannot outperform simple statistical methods due to several special properties of tabular data, including non-Gaussian distribution of continuous columns, the imbalanced distribution of discrete columns, and the mix of discrete and continuous columns. (See chapter 4 for these special properties.)

To address these challenges, we designed conditional tabular GAN (CTGAN), a method that introduces several new techniques. These include augmenting training procedures with reversible data transforms, architectural changes, and addressing data imbalance by employing a conditional generator. (See chapter 5 for model details.) When applied to the same datasets and evaluated with one benchmarking framework named SDGym, CTGAN performs significantly better than both the Bayesian network baselines and the other new GANs we tested.

We made several attempts before CTGAN: We developed a GAN-based model called TGAN, and a VAE-based model called TVAE. (See chapter 6 for details about TGAN and TVAE.) In designing these two models, we gained a lot of experience and better understood the challenges of this task, which helped us to design the better CTGAN model later.

The contributions of this project are summarized as follows:

- **Conditional GANs for synthetic data generation**. We propose CTGAN as a synthetic tabular data generator to address several issues in synthetic tabular data generation. We find CTGAN outperforms Bayesian networks and generates high-quality synthetic data.

- **Learning task and evaluation metrics**. Generating synthetic data is a complex task. We clearly define the learning task and the evaluation metric, then identify the challenges inherent in this task.

- **A benchmarking system for synthetic data generation algorithms**. We designed a comprehensive benchmark framework using several tabular datasets and different evaluation metrics, as well as implementations of several baselines and state-of-the-art methods. At the time of this writing, the benchmark has 5 deep learning methods, 2 Bayesian network methods, 15 datasets, and 2 evaluation mechanisms.

- **Other methods.** Before CTGAN, we designed different methods to generate synthetic data, namely TGAN and TVAE.

- **Opensource libraries.** We open-sourced CTGAN, SDGym and TGAN on GitHub. Upon writing, CTGAN has 27 stars and 7 forks; SDGym has 19 stars and 11 forks, and TGAN has 91 stars and 36 forks.

The rest of the thesis is organized as follows: Chapter 2 presents a primer of generative models; Chapter 3 describes the synthetic data generation task and four existing methods; Chapter 4 lists the challenges of this task; Chapter 6 introduces TGAN and TVAE; Chapter 5 introduces our model; Chapter 7 explains our benchmark design and implementation; and Chapter 8 and 9 gives our experimental results and conclusion.

Chapter 2

Primer on Generative Models

Generative models are machine learning models that attempt to learn the real data distribution and then draw samples from the learned distribution. It aligns with the objective of generating synthetic data. The generation of synthetic data is to learn the distribution of real data, and sample from the acquired distribution, while simultaneously satisfies other requirements such as privacy. Statistical generative models like Bayesian networks, Gaussian mixture models, and copulas are limited to a particular class of probability distributions, and can not model complicated distributions like images or text. In this chapter, we provide some background on deep generative models, namely variational autoencoder (VAE) and generative adversarial network (GAN) models, because of their effectiveness at modeling complex distributions like images and texts.

2.1 Variational autoencoders

An autoencoder has an encoder $\mathcal{E}(\cdot)$ and a decoder $\mathcal{D}(\cdot)$. Given a dataset $\mathbf{X}$, the encoder can encode an example the input data into a hidden distributed representation, denoted as

$$z = \mathcal{E}(x), \text{ where } x \sim \mathbf{X}.$$

The decoder can take this hidden representation and reconstruct the data

$$\hat{x} = \mathcal{D}(z), \text{ where } \hat{x} \approx x.$$

Autoencoders are typically trained with mean-squared error

$$\mathcal{L} = \mathbb{E}_{x \sim \mathbf{X}}\big[||\mathcal{D}(\mathcal{E}(x)) - x||_2^2\big]$$

In contrast to an autoencoder, a VAE [31] considers the hidden representation as a Gaussian distribution $\mathcal{N}(\cdot)$. So the encoder takes an example x and outputs a mean vector μ and a standard deviation vector σ

$$\mu, \sigma = \mathcal{E}(x), \text{ where } x \sim \mathbf{X},$$

meaning that the hidden representation follows $\mathcal{N}(\mu, \sigma^2)$. The decoder then takes one sample from the distribution and reconstructs the data

$$\hat{x} = \mathbb{E}_{z \sim \mathcal{N}(\mu,\sigma)}\big[\mathcal{D}(z)\big], \text{ where } \hat{x} \approx x.$$

VAE also constrains the aggregated distribution of z over all the data $\mathbf{X}$ to be $\mathcal{N}(\mathbf{0}, \mathbf{I})$. With this constraint, the VAE becomes a generative model because users can sample a random vector from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ and feed it to the decoder to generate data.

The learning objective of a VAE is the evidence lower-bound (ELBO) loss

$$\mathcal{L} = \mathbb{E}_{x \sim \mathbf{X}}\big[\underbrace{\mathbb{E}_{z \sim \mathcal{N}(\mu,\sigma\mathbf{I})}[||\mathcal{D}(z) - x||_2^2]}_{\text{Term I}} + \underbrace{\mathbb{KL}(\mathcal{N}(\mu, \sigma\mathbf{I})||\mathcal{N}(\mathbf{0}, \mathbf{I}))}_{\text{Term II}} \big]. \tag{2.1}$$

Term I is to train the model to work as an autoencoder. In Term II, $\mathbb{KL}(p||q)$ means the Kullback–Leibler (KL) divergence, which measures the distance between two distributions p and q as

$$\mathbb{KL}(p||q) = -\int_x p(x) \log \frac{q(x)}{p(x)}.$$

The close form of Term II is

$$-0.5 * ||1 + 2\log\sigma - \mu^2 - \sigma^2||_1$$

It constrains the distribution of z to be the standard normal distribution. When generating data, $\mathcal{D}(\cdot)$ takes $z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and outputs a synthetic sample. When the model converges, the learned generator is an approximated mapping from a multivariate Gaussian distribution to the data distribution.

An encoder and decoder are neural networks with multi-input and multi-output. The model is trained by stochastic gradient descent. The outer expectation is computed by taking the average over minibatch. The expectation in Term I of Eq. (2.1) is computed by sampling one z from the distribution.

2.2 Generative adversarial networks

Vanilla GAN: A vanilla GAN [17] has a generator $\mathcal{G}(\cdot)$ and a discriminator $\mathcal{D}(\cdot)$. The generator is supposed to project a multivariate Gaussian distribution to an arbitrary data distribution. The discriminator is supposed to tell whether the distribution parameterized by the generator is the same as the distribution of real training data. The generator and the discriminator are playing a zero-sum min-max game. When the generator and discriminator are perfect, they achieve a Nash equilibrium: The generator is modeling the exact data distribution and the discriminator cannot distinguish between these two distributions. When used for image generation, GANs can generate images that are perceptually[1] better than VAEs because

- The discriminator gives a better learning signal than the mean-square error. Discriminators can easily detect sharp and blurred images. If the training data is sharp and the generator generates blurred images, the discriminator can reject those images. If the generator generates a sharp realistic image, even if the image is far away from any image in the training set pixel-wise, the discriminator

[1] 'Perceptually' means a human cannot tell difference visually.

would accept the image. By doing so, the discriminator helps the generator generate perceptually realistic images. In contrast, the mean-squared error in VAEs encourages the decoder to output blurred images.

- In a complicated data space such as all images, a GAN is learning the exact probability distribution while a VAE is optimizing a lower-bound, thus learning an approximated distribution[31].

The generator $\mathcal{G}(\cdot)$ is a neural network that takes a random vector z from a $\mathcal{N}(\mathbf{0}, \mathbf{I})$, and projects z to a vector $\hat{x}$ in the data domain. The discriminator $\mathcal{D}(\cdot)$ is another neural network that predicts whether its input is from the real data distribution $\mathbf{X}$ or the learned distribution $\mathbb{P}_{\mathcal{G}}$. $\mathcal{D}(\cdot)$ outputs a continuous value in $[0, 1]$, where 0 means the input is from the learned distribution and 1 means the input is from the real data distribution. Therefore, $\mathcal{D}(\cdot)$ can be considered as a binary classifier. When training a GAN, the generator and discriminator are optimized iteratively. $\mathcal{D}(\cdot)$ is first optimized by

$$\mathcal{L}_{\mathcal{D}} = -\mathbb{E}_{x \sim \mathbf{X}}[\log \mathcal{D}(x)] + \mathbb{E}_{z \sim \mathcal{N}(\mathbf{0},\mathbf{I})}[\log(1 - \mathcal{D}(\mathcal{G}(z))].$$

$\mathcal{L}_{\mathcal{D}}$ is the cross-entropy loss for binary classification. Then $G(\cdot)$ is optimized by

$$\mathcal{L}_{\mathcal{G}} = \mathbb{E}_{z \sim \mathcal{N}(\mathbf{0},\mathbf{I})}[\log \mathcal{D}(\mathcal{G}(z)].$$

$\mathcal{L}_{\mathcal{G}}$ makes the generator maximize the chance of fooling the discriminator. When optimizing $\mathcal{L}_{\mathcal{G}}$, only the parameters in the $\mathcal{G}(\cdot)$ are optimized while $\mathcal{D}(\cdot)$ is fixed. Similarly, only the parameters in $\mathcal{D}(\cdot)$ are updated when optimizing $\mathcal{L}_{\mathcal{D}}$.

GAN is trained using stochastic gradient descent. In each iteration, there are two updates, one for the discriminator, the other for the generator.

- The discriminator is updated first. To do so, select a mini-batch of examples from training data, and a mini-batch of random vectors. First, input the random vectors to the generator and generate some fake data. Then feed both the real data and the fake data to the discriminator. Finally, compute $\mathcal{L}_{\mathcal{D}}$ and update parameters in the discriminator.

- The generator is updated afterward. To do so, select another mini-batch of random vectors. Use the generator to generate fake data. Input these fake data into the discriminator. Then compute $\mathcal{L}_{\mathcal{G}}$ and update parameters in the generator.

Vanilla GANs are hard to train. The model is sensitive to hyper-parameters and takes a long time to converge. When the learned distribution is far from the real data distribution – for example at the beginning of the training – the discriminator can only propagate small gradients to the generator, so the generator cannot learn effectively.

Wasserstein GAN: One recent improvement to GAN is the Wasserstein GAN (WGAN) [4]. Instead of using a discriminator, WGAN uses a critic network $\mathcal{C}(\cdot)$. The critic network outputs a larger value when the input is more realistic, and it outputs a smaller value when the input looks fake.

To achieve this, the critic network is trained by

$$\mathcal{L}_{\mathcal{C}} = -\mathbb{E}_{x \sim X}[\mathcal{C}(x)] + \mathbb{E}_{z \sim \mathcal{N}(\mathbf{0},\mathbf{I})}[\mathcal{C}(\mathcal{G}(z))]. \tag{2.2}$$

It tries to maximize the output on real data and minimize the output on fake data.

The generator is doing the opposite. It tries to maximize the output of the critic network. So the loss function for the generator is

$$\mathcal{L}_{\mathcal{G}} = -\mathbb{E}_{z \sim \mathcal{N}(\mathbf{0},\mathbf{I})}[\mathcal{C}(\mathcal{G}(z))].$$

WGAN is superior to vanilla GANs because WGAN actually minimizes the Wasserstein distance between the learned distribution and real data distribution. And Wasserstein distance has superior properties in that it can give reasonable gradients even when two distributions are far away, which benefits the training of GANs. Here we show how the loss function of WGAN is connected to Wasserstein distance.

Recall that the Wasserstein distance $\mathbb{W}(\cdot)$ can be computed using the Kantorovich-Rubinstein duality as

$$\mathbb{W}(\mathbf{X}, \mathbb{P}_{\mathcal{G}}) = \sup_{||f||_L \leq 1} \mathbb{E}_{x \sim \mathbf{X}}[f(x)] - \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\mathcal{G}}}[f(\hat{x})], \tag{2.3}$$

where $||f||_L \leq 1$ means f is a 1-Lipschitz function. Comparing Eq. (2.2) and Eq. (2.3), we observe that the $\mathcal{L}_{\mathcal{C}}$ is an approximation of $\mathbb{W}(\mathbf{X}, \mathbb{P}_{\mathcal{G}})$. The training of the critic is to make the approximation more accurate. In WGAN, the parameters in the critic are clipped to enforce 1-Lipschitz condition. The training of the generator is to minimize the Wasserstein distance.

WGAN is also trained by stochastic gradient descent and has two updates in each iteration.

- The critic is updated first. To update the critic network, sample a mini-batch of real data, and a mini-batch of random vectors. Use the generator to generate some fake data. Use the real data and fake data to compute $\mathcal{L}_{\mathcal{C}}$ and update $\mathcal{C}$. After the update, the parameters in $\mathcal{C}(\cdot)$ are clipped to a small range such as $[-0.1, 0.1]$ or $[-0.01, 0.01]$.

- The generator is updated afterward. Sample another mini-batch of random vectors, and generate fake data using the generator. Input the fake data to the critic network, then compute the $\mathcal{L}_{\mathcal{G}}$ to update the generator.

WGAN with gradient penalty: Clipping weights to enforce the 1-Lipschitz condition is not an optimal solution. An alternative solution is to enforce the 1-Lipschitz condition by constraining the gradients. The method is called WGANGP [18]. The idea stems from the fact that an optimal critic has a gradient norm equal to 1 almost everywhere on real data distribution and a learned distribution. WGANGP uses a regularization term that forces the norm of the gradient to be 1. The difference between WGANGP and WGAN is the critic loss function

$$\mathcal{L}_{\mathcal{C}} = \mathbb{E}_{x \sim \mathbf{X}, z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\mathcal{C}(x) - \mathcal{C}(\mathcal{G}(z)) + 10 \times (||\nabla \mathcal{C}(\tilde{x})||_2 - 1)^2 \right],$$

where $\tilde{x}$ is used to estimate the gradient and enforce the Lipschitz constraint. $\tilde{x}$ is a random linear composition of x and $\mathcal{G}(z)$ as

$$\tilde{x} = \rho x + (1 - \rho)\mathcal{G}(z), \text{ where } \rho \sim \mathcal{U}(0, 1).$$

$\mathcal{U}(0, 1)$ means a uniform distribution in $[0, 1]$. In WGANGP, clipping weights is no longer needed. WGANGP can generate more realistic data than GAN and WGAN.

In this research project, we explore the use of VAEs and GANs to generate synthetic data. In our `TVAE` model, we use VAEs to generate synthetic data. In our `CTGAN` model, WGANGP is used.

Chapter 3

Synthetic Data Generation Task and Related Work

In chapter 1, we highlighted the need for synthetic data. Synthetic data is a powerful tool that can overcome many barriers in data science. High-quality synthetic data can substitute for real data to alleviate privacy concerns. The quality of synthetic data is determined by whether the synthetic data correctly captures the correlations between different columns. In this chapter, we formally define the learning task so that the quality of synthetic data can be quantitatively evaluated. Due to the complexity of tabular data, we constrain our task to a specific type of tabular data, which is a single table with independent rows. We then explain a handful of models that fit our task.

3.1 Synthetic data generation

Because tabular data can take different forms, it can represent various types of information.

- **Single table with independent rows**: A table in this category can be thought of as several samples from an underlying joint distribution, where each row is sampled independently from the distribution. For example, a table of people's gender, height and weight fits in this category. Each row in the table is one sample from the joint distribution of gender, height, and weight of all human beings. Adding one row to the table does not affect the distribution of the other rows.

- **Single table with dependent rows**: In some tables, there are strong correlations between rows. For example, a table of gas stations and sales fits in this category. Assuming the consumption of gas is stable, adding one more gas station to the table is likely to reduce the sales of several nearby gas stations. Another very common example is time-series data. In time-series data, each row depends on previous rows.

- **Multiple-table database**: Sometimes, using multiple tables can represent information more effectively. Each table in a multi-table database can have dependent or independent rows. For example, a typical database for e-business includes a table of users, products, and transactions. The user and product tables have independent rows, while the transaction table is a time-series table containing dependent rows.

Since the research on generating synthetic data is in an early stage, we focus on the simplest case - a single table with independent rows.

Learning task definition: The synthetic data generation task is to train a data synthesizer G which takes a table as input and generate a synthetic version of this input. We require the input table to contain independent rows, and only continuous and discrete columns.[1] A table $\mathbf{T}$ contains N_c continuous columns $\{C_1, \ldots, C_{N_c}\}$, and N_d discrete columns $\{D_1, \ldots, D_{N_d}\}$. Each column is considered as a random variable. These random variables follow an unknown joint distribution $\mathbb{P}(C_{1:N_c}, D_{1:N_d})$. One row $\mathbf{r}_j = \{c_{1,j}, \ldots, c_{N_c,j}, d_{1,j}, \ldots, d_{N_d,j}\}$ is one sample from the joint distribution. $\mathbf{T}$ is then partitioned into training set $\mathbf{T}_{train}$ and test set $\mathbf{T}_{test}$. After training G on $\mathbf{T}_{train}$, $\mathbf{T}_{syn}$ is constructed by independently sampling rows from G.

Evaluation metrics: Direct evaluation of $\mathbf{T}_{syn}$ either tests whether $\mathbf{T}_{syn}$ and $\mathbf{T}_{train}$ are sampled from the same distribution or calculates the distance between two un-

[1] Continuous columns refer to ordinal columns that have many possible values; for example, integer or float columns. Discrete columns refer to categorical columns or ordinal columns with few values.

derlying distributions. Existing methods for this test make strong assumptions about the underlying distribution. For example, `Z-test` [10] assumes the data follows a Gaussian distribution. Due to these strong assumptions, these methods do not apply to tabular data with complicated distributions. Since direct evaluation is intractable, we use two alternative methods.

- **Sample likelihood:** We handcraft a table $\mathbf{T}$ for evaluation purposes. In this case, the underlying distribution of $\mathbf{T}_{train}$, denoted as $\mathbb{P}_{train}(\cdot)$, is known, and the underlying distribution of $\mathbf{T}_{syn}$, denoted as $\hat{\mathbb{P}}_{syn}(\cdot)$, can be approximated. The likelihood of $\mathbf{T}_{test}$ on $\hat{\mathbb{P}}_{syn}(\cdot)$, and the likelihood of $\mathbf{T}_{syn}$ on $\mathbb{P}_{train}(\cdot)$ can reveal the distance between two distributions.

- **Machine learning efficacy:** The previous metric requires underlying distributions of $\mathbf{T}_{train}$ and $\mathbf{T}_{syn}$ so that the likelihood can be computed. In more general cases, finding the underlying distributions is difficult. Alternatively, the quality of $\mathbf{T}_{syn}$ can be evaluated by machine learning applications such as classification or regression. For example, we can train a classifier or a regressor to predict one column using other columns as features. We can measure the efficacy by evaluating whether a classifier or regressor learned from $\mathbf{T}_{syn}$ can achieve equivalent or higher performance on $\mathbf{T}_{test}$ as a model learned on $\mathbf{T}_{train}$ would.

These two evaluation metrics are further elaborated in Chapter 7.

3.2 Existing techniques to generate synthetic data

The possibility of generating fully synthetic data appeals to different research communities, including statistics, database management, and machine learning. Earlier work in this area, such as `PrivBayes` [58], uses traditional Bayesian networks but adds a differentially private learning algorithm. Recently, GANs have been used in generating tabular data. GANs are appealing due to their performance and the flexibility they show in representing data, as evidenced by their success in generating and manipulating images and natural language [4, 18, 59, 57].

We surveyed a few recent developments published in the period 2017-2019. VeeGAN [50] uses GANs to generate 2D continuous data. [56] uses GANs to generate continuous time-series medical records. MedGAN [13] combines an autoencoder and a GAN to generate non-time-series continuous and/or binary data. [9] proposes to generate discrete tabular data using GAN. ehrGAN [11] generates augmented medical records, but doesn't explicitly generate synthetic data. TableGAN [39] tries to solve the problem of generating synthetic data using a convolutional neural network and explicitly optimizing the quality of the label column, thus their generated data can be used to train classifiers. PATE-GAN [30] tries to generate differentially private synthetic data. All GAN-based models mentioned above were published in 2017 and 2018. We find PrivBayes [58], MedGAN [13], TableGAN [39], and VeeGAN [50] are suitable for synthesizing a single table with independent rows. We introduce these 4 methods in this section[2].

3.2.1 PrivBayes

PrivBayes [58] aims to generate high-quality, differentially private [15] synthetic data using Bayesian networks.

Motivation: Bayesian networks can represent a joint distribution of discrete variables. Using a Bayesian network to generate differentially private synthetic data involves three steps: (1) learn a Bayesian network, (2) inject Laplace noise[3] to each parameter in the network, (3) sample from the noisy network. However, this process usually leads to low-quality synthetic data due to the large amount of noise injected in step (2). To achieve a certain privacy level, different network structures require different amounts of noise. PrivBayes proposes a heuristic to find a good network structure that needs less injected noise.

Preprocessing: Bayesian networks cannot model continuous variables. In PrivBayes,

[2] PATE-GAN is also suitable for our task. We do not explain PATE-GAN in detail because its contribution is based around differential privacy, while in this thesis we focus on the model architecture.

[3] The scale of the noise depends on the sensitivity [15] of the parameter and the privacy requirement.

all continuous variables are discretized into 16 equal-sized bins, so that the modeling algorithm only deals with discrete-valued columns.

Model details: There is a trade-off between the Bayesian network's original quality and the quality decrease after adding noise. For example, given a table with N_d discrete columns, a $(N_d - 1)$-way[4] Bayesian network can fit the distribution perfectly, but some weights in the network will have high sensitivity[5] and low value, meaning that noise can play a major role in a noisy model. Another extreme example involves using a 0-way network. The sensitivity is low, but because the model only learns the marginal distribution, it is not very useful. An appropriate number of ways is between 0 and $N_d - 1$. `PrivBayes` does not set a fixed number of ways. Instead, it uses a measure named θ-usefulness to balance the accuracy of the Bayesian network and the amount of noise needed. A noisy distribution is θ-useful if the ratio of average scale of information[6] to average scale of noise is no less than θ. Under the constraint of θ-usefulness, `PrivBayes` uses a greedy algorithm to find a graph that maximizes the mutual information.[7]

Datasets and evaluation metrics: `PrivBayes` is extensively evaluated on four real datasets. Machine learning efficacy is evaluated, as is the distance of marginal distribution. Since this method provides ϵ-differential privacy, the effect of ϵ on synthetic data quality is clearly shown in experiments.

Reproducibility: `PrivBayes` is implemented in a high-quality C++ code.[8] All the datasets are included in the package. We can reproduce all results reported in the paper using the code.

[4] A k-way Bayesian network means each node in the network can have at most k incoming edges; that is, each variable can be conditioned on at most k other variables.

[5] Sensitivity [15] of weight refers to how much the weight changes when one data row is removed. Sensitivity determines the amount of noise required for that weight to maintain a certain level of privacy.

[6] The scale of information is the reciprocal of the number of parameters to learn in the current step. For example, in a Bayesian network of all binary variables, if a node has k incoming edges, the parameters to learn are values in a $2^k \times 2$ table. Each row of the table represents one combination of k binary values, and each column represents one possible value of this node. The scale of information is $1/2^{k+1}$.

[7] Maximizing mutual information can find an optimal tree structure Bayesian network [14]. `PrivBayes` extends the algorithm from a tree to a graph.

[8] https://sourceforge.net/projects/privbayes/

3.2.2 MedGAN

Because health records are valuable for research but strictly protected for privacy reasons, healthcare is a domain that particularly desires synthetic data technology. To remove such barriers, `MedGAN` [13] uses a GAN framework to generate fully-synthetic health records.

Motivation: In health records, each column follows a very different distribution, making the learning of a GAN model difficult. Direct modeling cannot produce a good result. Thus in `MedGAN`, an autoencoder is deployed to project raw data into a lower-dimensional representation. After that, a GAN is used to generate such a representation.

Preprocessing: The model supports a table with all binary columns and all continuous columns. A binary column is simply represented as 0 or 1. A continuous variable is normalized to $[0, 1]$ using min-max normalization

$$\frac{c_{i,j} - \min(C_i)}{\max(C_i) - \min(C_i)}. \tag{3.1}$$

Model details: Figure 3-1 illustrates the model. In `MedGAN`, the generator and discriminator are working on different spaces. The generator generates a hidden representation. The discriminator checks raw data. So the output of the generator should go through the decoder before feeding into the discriminator, shown as the second workflow in Figure 3-1. During training, the autoencoder is trained first, shown as the first workflow in Figure 3-1. It is fixed when training the GAN. The loss function for the autoencoder is the mean squared error if the table contains only continuous columns, and cross-entropy loss if the columns are all binary. The generator and discriminator are trained using the same loss function as a vanilla GAN [17].

Datasets and evaluation metrics: Experiments are conducted on three electronic health records datasets. The machine learning efficacy is evaluated. Furthermore, the marginal distribution of each column is plotted and compared visually.

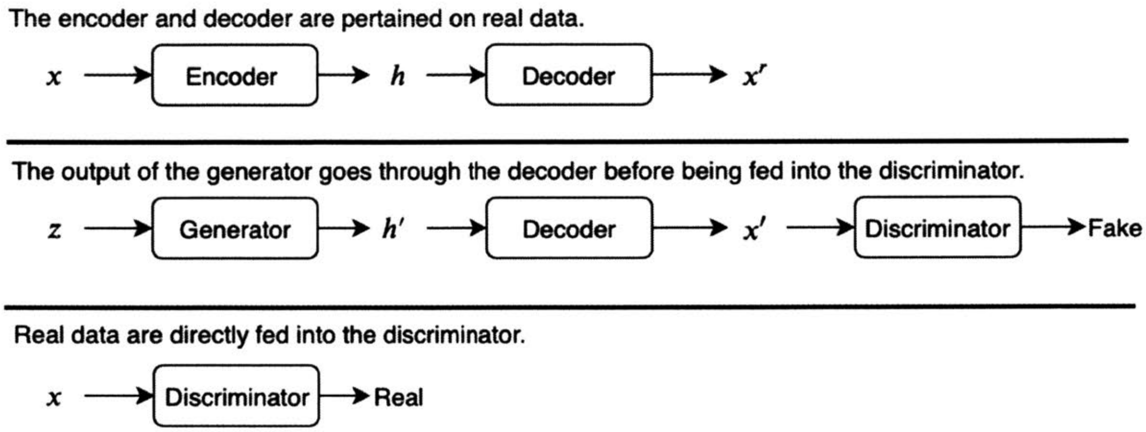

The encoder and decoder are pertained on real data.
x
Encoder
h
Decoder
x'
The output of the generator goes through the decoder before being fed into the discriminator.
z
Generator
h'
Decoder
x'
Discriminator
Fake
Real data are directly fed into the discriminator.
x
Discriminator
Real

Figure 3-1: The MedGAN framework contains an encoder, a decoder, a generator, and a discriminator. The encoder and decoder are pretrained on the real data and fixed in later steps. During training, the output of the generator is passed through the decoder before feeding into the discriminator. The discriminator checks whether the data are real or fake.

Reproducibility: MedGAN is implemented in Python and TensorFlow.[9] The original implementation does not support tables with mixed types. It only supports continuous and binary variables. I was not able to reproduce the results due to the availability of the data.

3.2.3 TableGAN

TableGAN [39] directly applies the idea of DCGAN [41] to generate synthetic data aimed at solving privacy issues.

Preprocessing: All continuous columns are normalized per Eq. (3.1). Discrete columns are also converted to a floating-point number. Each category in a discrete column is first represented by a unique integer in $\{0, \ldots, |D_i| - 1\}$ then divided by $|D_i| - 1$. Since DCGAN designed for images, the input is a matrix rather than a vector, and a row in a table is reshaped to a squared matrix. If the number of columns is not exactly a square number, zeros are padded to the row to increase the number of columns to the next square number. For example, if a table has 19 columns, the preprocessing method first appends 6 zero columns to the table, then reshapes each

[9] https://github.com/mp2893/medgan

row to a 5×5 matrix.

Model details: The model uses convolutional networks in both the generator and the discriminator, and is trained in the same way as a vanilla GAN. When tabular data contains a label column, a prediction loss is added to the generator to explicitly improve the correlation between the label column and other columns.

Datasets and evaluation metrics: The model is evaluated on four datasets. The evaluation metrics include machine learning efficacy and plots of marginal distributions.

Reproducibility: `TableGAN` is implemented in Python and TensorFlow.[10] The package includes two datasets used in the paper as well as scripts for preprocessing and training. We were not able to reproduce all results due to the availability of data.

3.2.4 VeeGAN

Although `VeeGAN` [50] is not designed to generate tabular data, it tackles the mode collapse issue, which turns out to be important for tabular data generation because continuous columns in tabular data have multiple modes. (See Chapter 4 for details about multi-modality in tabular data)

Motivation: Mode collapse is a known issue for GANs. It stems from the fact that GANs are reluctant to generate certain objects in an image, and such a flaw is noticeable when data has lower dimensionality (See Figure 4-2). To address this problem, `VeeGAN` introduces a reconstructor module that can detect when mode collapse occurs and guide the generator to overcome the flaw.

Preprocessing: `VeeGAN` is designed for continuous data. All values are min-max normalized to $[0, 1]$ as Eq. (3.1).

Model details: In `VeeGAN`, the reconstructor does the opposite of the generator, and projects a row back to the random vector. With a generator, a discriminator, and a reconstructor, we can construct two workflows, as shown in Figure 3-2. The three modules are trained as follows:

[10] https://github.com/mahmoodm2/tableGAN

- Generator: The first step of the loss function is to fool the discriminator, the same as would happen with a vanilla GAN [17]. The second step is to minimize the L2 distance between z and z^r.
- Discriminator: The discriminator is trained as a binary classifier using cross-entropy loss.
- Reconstructor: The reconstructor also minimizes the distance between z and z^r.

Figure 3-3 illustrates how a reconstructor helps with mode collapse. The generator captures one of two modes and projects z_0 to x_2. But the reconstructor can project any point in the data space to the noise space. Data from the left mode are also projected to the same space as the right mode, causing overlaps after projection. For example, x_1 and x_2 are both projected to z_0. At this time, x_1 and x_2 are connected by the reconstructor. (x_1, z_0) is more likely to be classified as real by the discriminator because the discriminator never sees a negative example from the left mode. When the generator generates x_2 from z_0, the gradient will guide it to generate x_1 to fool the discriminator, eliminating the mode collapse.

Datasets and evaluation metrics: The model is evaluated on two 2D datasets. The evaluation metric is the number of modes captured by GAN which can be observed visually.

Reproducibility: `VeeGAN` is implemented in Python and Tensorflow.[11] The implementation is inconsistent with the descriptions in the paper. It also uses undocumented APIs in Tensorflow, making the code difficult to understand. After trying the model, we failed to reproduce the results.

[11] https://github.com/akashgit/VEEGAN

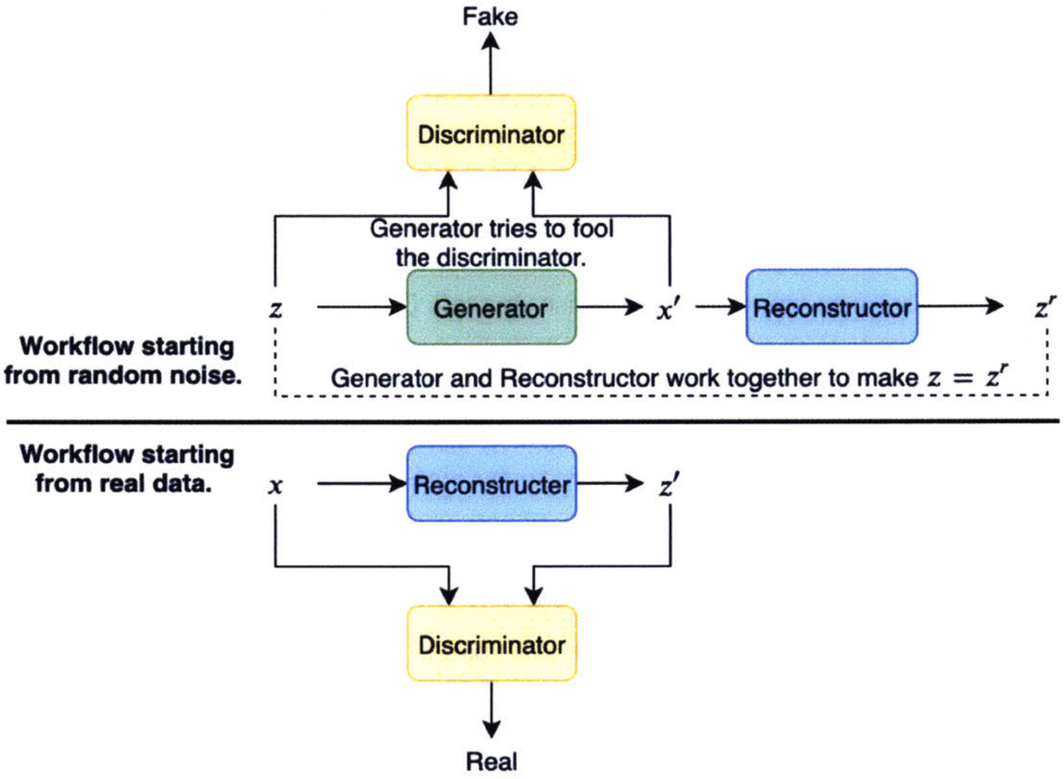

Fake
Discriminator
Generator tries to fool the discriminator.
z
Generator
x'
Reconstructor
z^r
Workflow starting from random noise.
Generator and Reconstructor work together to make z = z^r
Workflow starting from real data.
x
Reconstructer
z'
Discriminator
Real

Figure 3-2: VeeGAN framework. VeeGAN contains three modules: a generator, a discriminator, and a reconstructor. The top section shows the workflow starting from random noise z. In this workflow, the generator projects z to synthetic data x' and tries to fool the discriminator. The gradients from the discriminator help the generator improve. Meanwhile, the reconstructor learns to project x' back to $z^r = z$. The bottom section shows the workflow starting from real data x. x is inputted to the reconstructor in order to generate a representation vector z'. Then real tuples (x, z') and fake tuples (x', z) are used to train the discriminator.

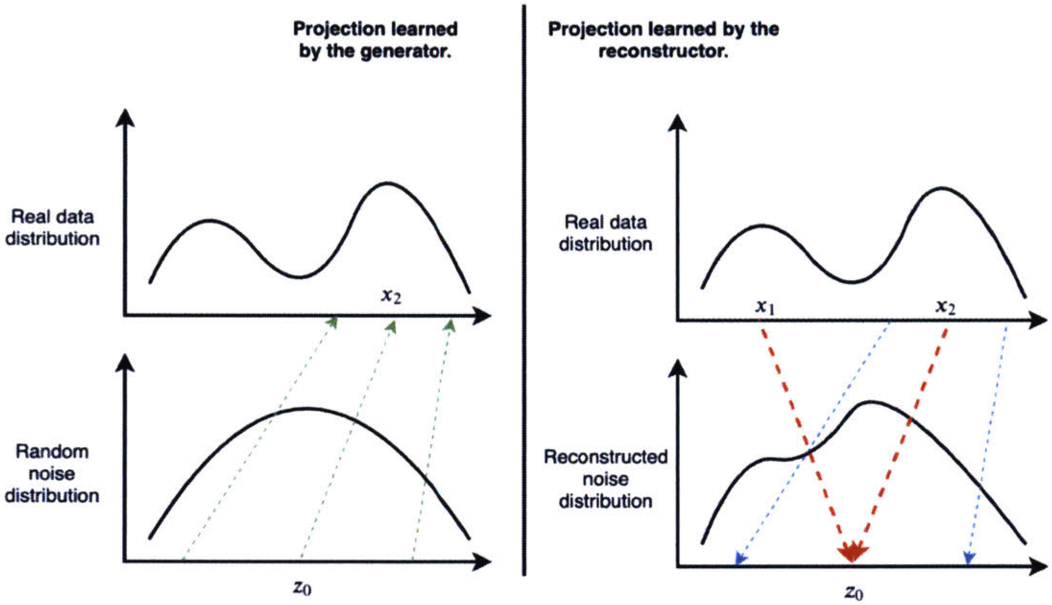

Projection learned by the generator.
Real data distribution
x_2
Random noise distribution
z_0
Projection learned by the reconstructor.
Real data distribution
x_1
x_2
Reconstructed noise distribution
z_0

Figure 3-3: The reconstructor makes a connection between missing modes and existing modes, so that the generator can recover from mode collapse. The left section shows how the generator projects random noise into the data space. The right section shows how the reconstructor projects data into the noise space.

Chapter 4

Challenges of Modeling Tabular Data using GAN

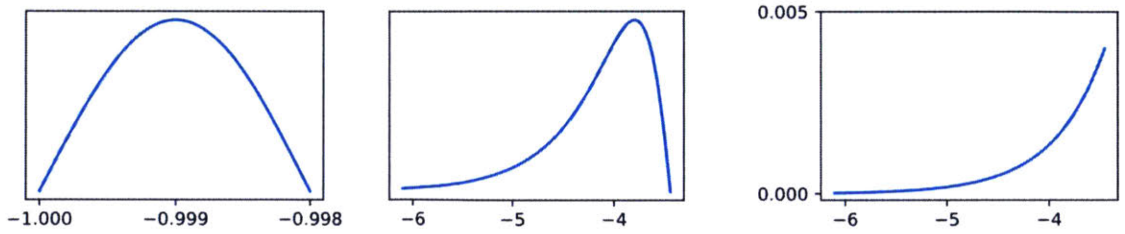
Several unique properties of tabular data make designing a GAN-based model challenging. In this section, we first highlight these challenges as they relate to single table non-time series data, which we try to address in our model. We then summarize other challenging properties of time series data and multiple-table data.

4.1 Challenges on single-table non-time-series data

Modeling and synthetically generating single-table non-time series data is the simplest problem in synthetic data. Each row of in the table is sampled independently from the distribution of all possible rows. One could argue that if a row of data is represented as a vector, specifically using min-max normalization on continuous values and one-hot representation for discrete values, then GAN models designed for images could easily be adapted to tabular data. However, here we list several special properties of single-table non-time-series data that can break this naive adaptation.

C1. Mixed data types. Real-world tabular data consists of mixed data types (continuous, ordinal, categorical, etc.). Each column has a complicated correlation with other columns. Traditional GAN for images uses `sigmoid` activation to generate each pixel. For tabular data, modifications to GANs must apply both `softmax` and `tanh` on the output to simultaneously generate a mix of discrete and continuous columns. Meanwhile, the modeling technique should be able to model the probability density of mixed discrete-continuous distribution.

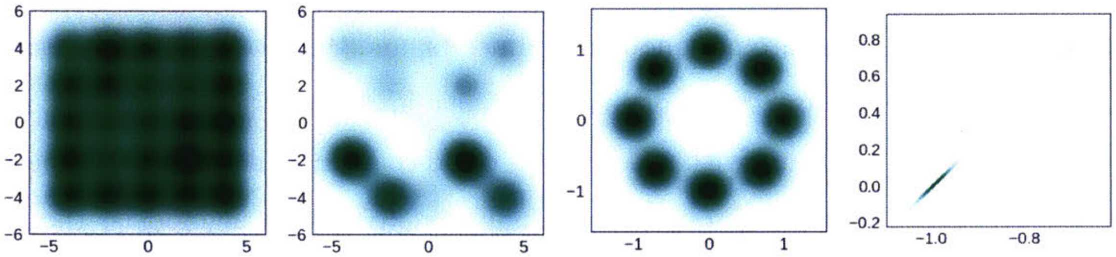
C2. Non-Gaussian distributions: In images, a pixel's values follow a Gaussian-like distribution, which can be normalized to $[-1, 1]$ using a min-max transform. A `tanh` function is usually employed in the last layer of a network to output a value in this range. Continuous variables in tabular data are usually non-Gaussian and have distributions with long tails; thus, most generated values will not be centered around zero. Very likely, the gradient of `tanh` where most values will be located is flat – a phenomenon known as gradient saturation. This results in the model's inability to learn via gradients. For example, annual household income has a long-tail distribution. The average of this column is around \$60k. However, there are several outliers that make more than \$100M a year. Applying min-max normalization would be problematic in this scenario because most of the values would be squeezed to $[-1, -0.998]$. In this range, the gradient of `tanh` vanishes. In Figure 4-1, we visualize this scenario. A vanished gradient not only prevents the model from learning the distribution of a column effectively. It also lets other columns with larger gradients occupy most of the model capacity.



(A) (B) (C)

Figure 4-1: Challenges of non-Gaussian distribution on GAN models. Assume we have a non-Gaussian continuous column with a few large outliers. The large outliers squeeze all other values towards -1. After min-max normalization, the probability density looks like (A). To use a neural network with `tanh` activation to generate this column, the probability distribution of values before the `tanh` activation looks like (B). The gradient of tanh vanishes in this range, as shown in (C). The model can not learn effectively with a vanished gradient.

C3. Multimodal distributions. Continuous columns in tabular data usually have multiple modes. We observe that 57/123 continuous columns in our 8 real-world datasets have multiple modes. [50] showed that a vanilla GAN couldn't model all modes on a simple 2D dataset as illustrated in Figure 4-2; thus it also wouldn't be able to model the multimodal distribution of continuous columns. This is a known issue of vanilla GAN [41, 4, 18, 36]. Vanilla GANs make their real/fake decision on only one example, so if the generator figures out one realistic example and tries to repeat that example every time, the discriminator does not have enough information to figure out the issue.



(A) (B) (C) (D)

Figure 4-2: [50] show that a vanilla GAN can not model a simple 2-dimensional Gaussian mixture. (A) is the probability density of 25 Gaussian distributions aligned as a grid. (B) is the corresponding distribution learned by GAN. (C) is the original distribution and (D) is the corresponding distribution learned by GAN.

C4. Learning from sparse one-hot-encoded vectors. To enable learning from non-ordinal categorical columns, a categorical column is converted into a one-hot vector. When generating synthetic samples, a generative model is trained to generate a probability distribution over all categories using `softmax`. This is problematic in GANs because a trivial discriminator can simply distinguish real and fake data by checking the distribution's sparseness instead of considering the overall realness of a row.

C5. Highly imbalanced categorical columns. In real-world datasets, most categorical columns have a highly imbalanced distribution. In our datasets, we noticed that 636/1048 of the categorical columns are highly imbalanced – the major category appears in more than 90% of the rows, creating severe mode collapse. Missing a minor category only causes tiny changes to the data distribution, but imbalanced data leads to insufficient training opportunities for minor classes. The critic network cannot detect such issues unless mode-collapse-preventing mechanisms such as `PacGAN` are used. These mechanisms can prevent GANs from generating only the most salient category. Synthetic data for minor categories are expected to be of lower quality, necessitating resampling.

C6. High dimensionality. The high dimensionality of tabular data increases the complexity exponentially. For example, n binary variables have 2^n possibilities. Accurately representing the probability distribution using a small neural network is impossible because there are not enough parameters, and there is usually not enough training data. In this case, any modeling technique introduces bias to the estimate. For example, when modeling with a GAN, bias could come in while choosing a specific network structure or learning objective. Compared to statistical models, the bias introduced in neural network models is hard to analyze.

C7. Lack of training data. Learning with small training data is a challenging problem. Similar problems have been branded as *few-shot learning* or *meta learning.* Such tasks are easier with images because content in different images shares similar filters. However, tabular data is drastically different. It is challenging to effectively transfer knowledge learned from one table to another.

C8. Missing values. Tabular data has missing values. To directly train a GAN model on tabular data with missing values, one should modify the data representation to properly distinguish missing values from known values, and mask the model to make it robust towards missing values. An alternative approach is to impute the missing values before training the model. However, the data imputation also requires modeling of the table. Mistakes in data imputation would be propagated to learned GAN models.

Table 4.1 shows whether existing methods explicitly address these challenges.

Table 4.1: A summary showing whether existing methods and our CTGAN explicitly address the aforementioned challenges [C1 - C8]. (* indicates it is able to model continuous and binary.)

Problems	MedGAN	TableGAN	PATE-GAN	CTGAN
C1	✓*	✓*	✓*	✓
C2	x	x	x	✓
C3	x	✓	x	✓
C4	x	x	x	✓
C5	x	✓	x	✓
C6	✓	✓	✓	✓
C7	x	x	x	x
C8	x	x	x	x

4.2 Challenges on time-series data

Time series data is another type of tabular data. The major difference between time series and non-time series data is that the distribution of each row is no longer independent. In time-series data, the distribution of a row is conditioned on previous rows. Modeling time series data requires the model to be capable of modeling conditional distribution.

4.3 Challenges on multiple table data

Data with complicated information is stored as multiple tables in a relational database. Modeling this type of data raises more challenges for GAN models. In multiple table data, there could be several non-time series tables and several time series tables. For example, a typical e-business website could have a table of users, a table of products, and a table of orders. The user table and the product table are non-time series, while the order table is time series. Modeling such distributions is still challenging, and would require the integration of heuristic algorithms such as SDV [40] with different GAN models.

Chapter 5

Conditional Tabular GAN

The failure of existing methods has emphasized the necessity of designing a new method to generate high-quality synthetic data. We find that a proper preprocessing method and an improved learning algorithm are required to make GANs work on tabular data. In this chapter, we explain our `CTGAN` model.

5.1 Notations

We define the following notations.

- $x_1 \oplus x_2 \oplus \ldots$: concatenate vectors $x_1, x_2, \ldots$
- $\texttt{gumbel}_\tau(x)$: apply Gumbel softmax[27] with parameter τ on a vector x
- $\texttt{leaky}_\gamma(x)$: apply a leaky ReLU activation on x with leaky ratio γ
- $\texttt{FC}_{u\to v}(x)$: apply a linear transformation on a u-dim input to get a v-dim output.

We also use `tanh`, `ReLU`, `softmax`, `BN` for batch normalization [25], and `drop` for dropout [51].

5.2 Mode-specific normalization

Properly representing the data is critical for training neural networks. Discrete values can naturally be represented as one-hot vectors, while representing continuous values with arbitrary distribution is non-trivial. Previous models [13, 39] use min-max normalization to normalize continuous values to $[-1, 1]$. In CTGAN, we design a *mode-specific* normalization to deal with columns with complicated multi-modal distributions.

Figure 5-1 shows our mode-specific normalization for a continuous column. In our method, each column is processed independently. Each value is represented as a one-hot vector indicating the mode, and a scalar indicating the value within the mode. Our method contains three steps.

1. For each continuous column C_i, use variational Gaussian mixture model (VGM) [7] to estimate the number of modes m_i and fit a Gaussian mixture. For instance, in Figure 5-1, the VGM finds three modes ($m_i = 3$), namely η_1, η_2 and η_3. The learned Gaussian mixture is $\mathbb{P}_{C_i}(c_{i,j}) = \sum_{k=1}^{3} \mu_k \mathcal{N}(c_{i,j}; \eta_k, \phi_k)$ where μ_k and ϕ_k are the weight and standard deviation of a mode respectively.

2. For each value $c_{i,j}$ in C_i, compute the probability of $c_{i,j}$ coming from each mode. For instance, in Figure 5-1, the probability densities are ρ_1, ρ_2, ρ_3. The probability densities are computed as $\rho_k = \mu_k \mathcal{N}(c_{i,j}; \eta_k, \phi_k)$.

3. Sample one mode from the given probability density, and use the sampled mode to normalize the value. For example, in Figure 5-1, we pick the third mode given ρ_1, ρ_2 and ρ_3. Then we represent $c_{i,j}$ as a one-hot vector $\beta_{i,j} = [0, 0, 1]$ indicating the third mode, and a scalar $\alpha_{i,j} = \frac{c_{i,j} - \eta_3}{4\phi_3}$ to represent the value within the mode.

The representation of a row becomes the concatenation of continuous and discrete columns

$$\mathbf{r}_j = \alpha_{1,j} \oplus \beta_{1,j} \oplus \ldots \oplus \alpha_{N_c,j} \oplus \beta_{N_c,j} \oplus \mathbf{d}_{1,j} \oplus \ldots \oplus \mathbf{d}_{N_d,j},$$

where $\mathbf{d}_{i,j}$ is a one-hot representation of a discrete value.

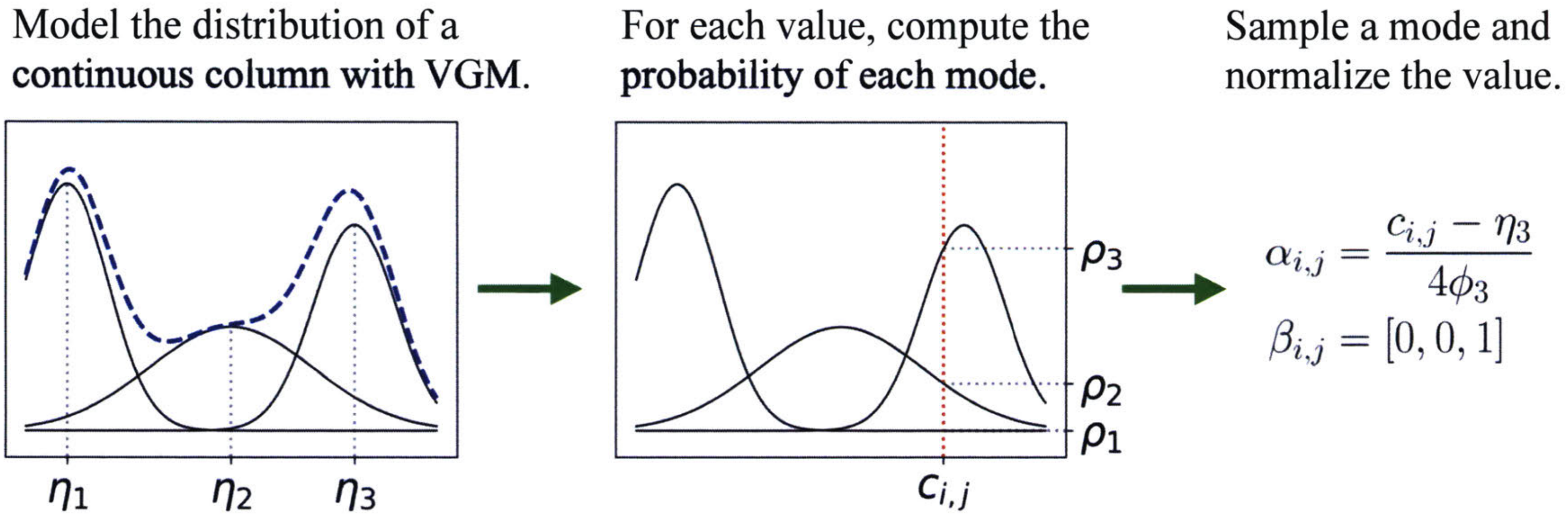

Model the distribution of a **continuous column with VGM.**
For each value, compute the **probability of each mode.**
Sample a mode and normalize the value.
η_1
η_2
η_3
ρ_3
ρ_2
ρ_1
$c_{i,j}$
$\alpha_{i,j} = \frac{c_{i,j} - \eta_3}{4\phi_3}$
$\beta_{i,j} = [0, 0, 1]$

Figure 5-1: An example of mode-specific normalization. The distribution of a continuous column (the blue dashed line in the left figure) has 3 modes, and these modes are modeled by a variational Gaussian mixture model. In the middle figure, a value from that column $c_{i,j}$ appears. $c_{i,j}$ has the probability density of ρ_1, ρ_2, ρ_3 of coming from each mode. It is more likely to come from the third mode. So $c_{i,j}$ is normalized by the mean and standard deviation of the third mode, namely η_3 and ϕ_3.

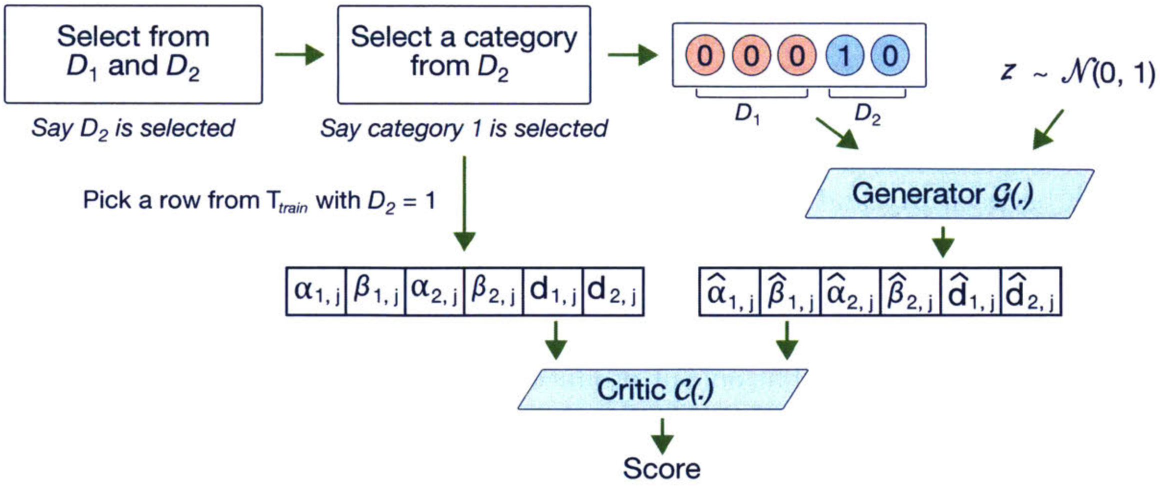
5.3 Conditional tabular GAN architecture

Traditionally, a GAN is fed with a vector sampled from a standard multivariate normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, and by means of the *Generator* and *Discriminator* or *Critic*([4],[18]) neural networks, one eventually obtains a deterministic transformation that maps $\mathcal{N}(\mathbf{0}, \mathbf{I})$ onto the distribution of the data.

This method of training a generator does not account for the imbalance in the categorical columns. If the training data are randomly sampled during training, the rows that fall into the minor category will not be sufficiently represented, and the generator may not be trained correctly. If the training data are resampled, the generator learns the resampled distribution, which is different from the real data distribution. This problem is reminiscent of the "*class imbalance*" problem in discriminatory modeling, but the challenge is exacerbated, since there is not a single column to balance and the real data distribution should be kept intact.

Specifically, the goal is to resample efficiently so that all the categories from discrete attributes are sampled evenly (but not necessarily uniformly) during the training process, and to recover the (not-resampled) real data distribution during the test or synthesis. One way to attain this is to enforce that the generator matches a given category. Let k^* be the value from the i^*th discrete column D_{i^*} that has to be matched by the generated samples $\hat{\mathbf{r}}$, then the generator can be interpreted as the conditional distribution of rows given that particular value at that particular column, i.e. $\hat{\mathbf{r}} \sim \mathbb{P}_{\mathcal{G}}(\text{row}|D_{i*} = k^*)$. For this reason, we call this generator a *Conditional generator*, and a GAN built upon it is referred to as a *Conditional GAN*. Moreover, in this thesis we construct our `CTGAN` as a Conditional GAN upon two main modules: the conditional generator $\mathcal{G}$ and the critic $\mathcal{C}$.

Integrating a conditional generator into the architecture of a GAN requires tackling the following issues: 1) it is necessary to devise a representation for the condition as well as to prepare an input for it, 2) it is necessary for the generated rows to preserve the condition as it is given, and 3) it is necessary for the conditional generator to learn the real data conditional distribution, i.e. $\mathbb{P}_{\mathcal{G}}(\text{row}|D_{i*} = k^*) = \mathbb{P}(\text{row}|D_{i*} = k^*)$,



Select from D_1 and D_2
Say D_2 is selected
Select a category from D_2
Say category 1 is selected
0 0 0 1 0
D_1
D_2
$z \sim \mathcal{N}(0, 1)$
Pick a row from T_{train} with $D_2 = 1$
Generator $\mathcal{G}(.)$
$\alpha_{1,j}$ $\beta_{1,j}$ $\alpha_{2,j}$ $\beta_{2,j}$ $d_{1,j}$ $d_{2,j}$
$\hat{\alpha}_{1,j}$ $\hat{\beta}_{1,j}$ $\hat{\alpha}_{2,j}$ $\hat{\beta}_{2,j}$ $\hat{d}_{1,j}$ $\hat{d}_{2,j}$
Critic $\mathcal{C}(.)$
Score

Figure 5-2: CTGAN structure.

so that

$$\mathbb{P}(\text{row}) = \sum_{k \in D_{i*}} \mathbb{P}_{\mathcal{G}}(\text{row}|D_{i*} = k^*)\mathbb{P}(D_{i^*} = k).$$

We present a solution that consists of three key elements, namely: the *conditional vector*, the generator loss, and the *training-by-sampling* method.

Conditional vector. We introduce the vector *cond* to indicate the condition ($D_{i^*} = k^*$). Recall that after the reversible data transformation, all the discrete columns $D_1, \ldots, D_{N_d}$ end up as one-hot vectors $\mathbf{d}_1, \ldots, \mathbf{d}_{N_d}$ such that the ith one-hot vector is $\mathbf{d}_i = [\mathbf{d}_i^{(k)}]$, for $k = 1, \ldots, |D_i|$. Let $\mathbf{m}_i = [\mathbf{m}_i^{(k)}]$, for $k = 1, \ldots, |D_i|$ be the ith *mask* vector associated to the ith one-hot vector $\mathbf{d}_i$. Hence, the condition can be expressed in terms of these mask vectors as

$$\mathbf{m}_i^{(k)} = \begin{cases} 1 & \text{if} \quad i = i^* \quad \text{and} \quad k = k^*, \\ 0 & \text{otherwise.} \end{cases}$$

Then, define the vector *cond* as *cond* $= \mathbf{m}_1 \oplus \ldots \oplus \mathbf{m}_{N_d}$. For instance, for two discrete columns, $D_1 = \{1, 2, 3\}$ and $D_2 = \{1, 2\}$,the condition ($D_2 = 1$) is expressed by the mask vectors $\mathbf{m}_1 = [0, 0, 0]$ and $\mathbf{m}_2 = [1, 0]$; so *cond* $= [0, 0, 0, 1, 0]$. ([0, 0, 0] means no value is assigned to D_1.)

Generator loss. During training, the conditional generator is free to produce any set of one-hot discrete vectors $\{\hat{\mathbf{d}}_1, \ldots, \hat{\mathbf{d}}_{N_d}\}$. In particular, given the condition ($D_{i^*} = k^*$)

in the form of *cond* vector, nothing in the feed-forward pass prevents from producing either $\hat{\mathbf{d}}_{i^*}^{(k^*)} = 0$ or $\hat{\mathbf{d}}_{i^*}^{(k)} = 1$ for $k \neq k^*$. The mechanism proposed to enforce the conditional generator to produce $\hat{\mathbf{d}}_{i^*} = \mathbf{m}_{i^*}$ is to penalize its loss by adding the cross-entropy between $\mathbf{m}_{i^*}$ and $\hat{\mathbf{d}}_{i^*}$, averaged over all the instances of the batch. Thus, as the training advances, the generator learns to make an exact copy of the given $\mathbf{m}_{i^*}$ into $\hat{\mathbf{d}}_{i^*}$.

Training-by-sampling. The output produced by the conditional generator must be assessed by the critic, which estimates the distance between the learned conditional distribution $\mathbb{P}_{\mathcal{G}}(\text{row}|cond)$ and the conditional distribution on real data $\mathbb{P}(\text{row}|cond)$. The sampling of real training data and the construction of the *cond* vector should comply to help the critic estimate the distance. There are two possibilities: either we randomly select an instance (row) from the table and then select the condition attribute within it, or we randomly select an attribute (column) and a value from that column and then select a row filtering the table by the value of that column. The first possibility is not appropriate for our goal because we cannot ensure that all the values from discrete attributes are sampled evenly during the training process. On the other hand, if we consider all the discrete columns equally likely and randomly select one, and then consider all the values in its range equally likely, it might be the case that one row from a very low-frequency category will be excessively oversampled; so once again is not an appropriate choice. Thus, for our purposes, we propose the following steps:

1. Create N_d zero-filled mask vectors $\mathbf{m}_i = [\mathbf{m}_i^{(k)}]_{k=1\ldots|D_i|}$, for $i = 1, \ldots, N_d$, so the ith mask vector corresponds to the ith column, and each component is associated with the category of that column.

2. Randomly select a discrete column D_i out of all the N_d discrete columns, with equal probability. Let i^* be the index of the column selected. For instance, in Figure 5-2, the selected column was D_2, so $i^* = 2$.

3. Construct a PMF across the range of values of the column selected in 2, D_{i^*}, such that the probability mass of each value is the logarithm of its frequency in

that column.

4. Let k^* be a randomly selected value according to the PMF above. For instance, in Figure 5-2, the range D_2 has two values and the first one was selected, so $k^* = 1$.

5. Set the k^*th component of the i^*th mask to one, i.e. $\mathbf{m}_{i^*}^{(k^*)} = 1$.

6. Calculate the vector $cond = \mathbf{m}_1 \oplus \cdots \mathbf{m}_{i^*} \oplus \mathbf{m}_{N_d}$. For instance, in Figure 5-2, we have the masks $\mathbf{m}_1 = [0, 0, 0]$ and $\mathbf{m}_{2*} = [1, 0]$, so $cond = [0, 0, 0, 1, 0]$.

We use the PacGAN framework, taking 10 samples from the training data in each pac. The training algorithm under this framework is completely described in Algorithm 1. It begins by creating as many condition vectors *cond*, and drawing as many samples from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, as the batch size (lines 1-3). Both are fed-forward into the conditional generator to produce a batch of *fake* rows (line 4). The input to PacGAN is twofold. On the one hand, it comes from sampling the training tabular data according to the *cond* vector. On the other hand, it is the output of the conditional generator. Both are preprocessed as detailed in lines 7 and 8 before being fed-forwarded into the critic, to obtain its loss $\mathcal{L}_C$ (line 9). In lines 10-12 we follow [18] to compute the gradient penalty for the critic. To update the parameters of the critic we use a gradient descent step, with learning rate $2 \cdot 10^{-4}, \beta_1 = 0.9, \beta_2 = 0.5$ and Adam optimizer (line 13). In order to update the parameters of the conditional generator, it is first necessary to repeat the feed-forward steps both in the conditional generator (lines 1-7) and in the critic (line 15). This leads to the loss of the conditional generator, since in this step the critic is not updated. Then, we use a gradient descent step similar to the one for the parameters of the critic (line 16).

Finally, the conditional generator $\mathcal{G}(z, cond)$ architecture can be formally described as

$$\begin{cases} h_1 = \texttt{ReLU}(\texttt{BN}(\texttt{FC}_{|cond|+|z|\to 256}(z \oplus cond))) & \\ h_2 = \texttt{ReLU}(\texttt{BN}(\texttt{FC}_{|cond|+|z|+256\to 256}(z \oplus cond \oplus h_1))) & \\ \hat{\alpha}_i = \texttt{tanh}(\texttt{FC}_{|cond|+|z|+512\to 1}(h_2)) & 1 \leq i \leq N_c \\ \hat{\beta}_i = \texttt{gumbel}_{0.2}(\texttt{FC}_{|cond|+|z|+512\to m_i}(h_2)) & 1 \leq i \leq N_c \\ \hat{\mathbf{d}}_i = \texttt{gumbel}_{0.2}(\texttt{FC}_{|cond|+|z|+512\to |D_i|}(h_2)) & 1 \leq i \leq N_d \end{cases}$$

and the architecture of the critic (with pac size 10) $\mathcal{C}(\mathbf{r}_1, \ldots, \mathbf{r}_{10}, cond_1, \ldots, cond_{10})$ can be formally described as

$$\begin{cases} h_0 = \mathbf{r}_1 \oplus \ldots \oplus \mathbf{r}_{10} \oplus cond_1 \oplus \ldots \oplus cond_{10} \\ h_1 = \texttt{drop}(\texttt{leaky}_{0.2}(\texttt{FC}_{10|\mathbf{r}|+10|cond|\to 256}(h_0))) \\ h_2 = \texttt{drop}(\texttt{leaky}_{0.2}(\texttt{FC}_{256\to 256}(h_1))) \\ \mathcal{C}(\cdot) = \texttt{FC}_{256\to 1}(h_2) \end{cases}$$

Generate synthetic data for different purposes. During testing, the user has to provide the Conditional CTGAN both with a random vector z (as to any other GAN) and a *cond* vector properly constructed according to the discrete columns and their range of values. Users can construct *cond* to generate rows with a specific value in a discrete column, for example generating several columns with $D_2 = 1$. In our experiments, i^* is sampled uniformly and $\mathbf{m}_{i^*}$ follows the marginal distribution of D_{i^*} so that the generated data are expected to reveal the real data distribution.

Algorithm 1: Train `CTGAN` on step.

```
Input: Training data $\mathbf{T}_{train}$, Conditional generator and Critic parameters $\Phi_G$ and $\Phi_C$ respectively, batch size $m$, pac size $pac$.
Result: Conditional generator and Critic parameters $\Phi_G$, $\Phi_C$ updated.
1  Create masks $\{\mathbf{m}_1, \dots, \mathbf{m}_{i^*}, \dots, \mathbf{m}_{N_d}\}_j$,   for $1 \leq j \leq m$
2  Create condition vectors $cond_j$,   for $1 \leq j \leq m$ from masks      ▷ Create $m$ conditional vectors
3  Sample $\{z_j\} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ,   for $1 \leq j \leq m$
4  $\hat{\mathbf{r}}_j \leftarrow \texttt{Generator}(z_j, cond_j)$ ,   for $1 \leq j \leq m$      ▷ Generate fake data
5  Sample $\mathbf{r}_j \sim \texttt{Uniform}(\mathbf{T}_{train} | cond_j)$ ,   for $1 \leq j \leq m$      ▷ Get real data

6  $cond_k^{(pac)} \leftarrow cond_{k \times pac+1} \oplus \dots \oplus cond_{k \times pac+pac}$,   for $1 \leq k \leq m/pac$      ▷ Conditional vector pacs
7  $\hat{\mathbf{r}}_k^{(pac)} \leftarrow \hat{\mathbf{r}}_{k \times pac+1} \oplus \dots \oplus \hat{\mathbf{r}}_{k \times pac+pac}$ ,   for $1 \leq k \leq m/pac$      ▷ Fake data pacs
8  $\mathbf{r}_k^{(pac)} \leftarrow \mathbf{r}_{k \times pac+1} \oplus \dots \oplus \mathbf{r}_{k \times pac+pac}$ ,   for $1 \leq k \leq m/pac$      ▷ Real data pacs
9  $\mathcal{L}_C \leftarrow \frac{1}{m/pac} \sum_{k=1}^{m/pac} \texttt{Critic}(\hat{\mathbf{r}}_k^{(pac)}, cond_k^{(pac)}) - \frac{1}{m/pac} \sum_{k=1}^{m/pac} \texttt{Critic}(\mathbf{r}_k^{(pac)}, cond_k^{(pac)})$

10 Sample $\rho_1, \dots, \rho_{m/pac} \sim \texttt{Uniform}(0, 1)$
11 $\tilde{\mathbf{r}}_k^{(pac)} \leftarrow \rho_k \hat{\mathbf{r}}_k^{(pac)} + (1 - \rho_k) \mathbf{r}_k^{(pac)}$ ,   for $1 \leq k \leq m/pac$
12 $\mathcal{L}_{GP} \leftarrow \frac{1}{m/pac} \sum_{k=1}^{m/pac} (||\nabla_{\tilde{\mathbf{r}}_k^{(pac)}} \texttt{Critic}(\tilde{\mathbf{r}}_k^{(pac)}, cond_k^{(pac)})||_2 - 1)^2$      ▷ Gradient Penalty [18]
13 $\Phi_C \leftarrow \Phi_C - 0.0002 \times \texttt{Adam}(\nabla_{\Phi_C}(\mathcal{L}_C + 10\mathcal{L}_{GP}))$

14 Regenerate $\hat{\mathbf{r}}_j$ following lines 1 to 7
15 $\mathcal{L}_G \leftarrow -\frac{1}{m/pac} \sum_{k=1}^{m/pac} \texttt{Critic}(\hat{\mathbf{r}}_k^{(pac)}, cond_k^{(pac)}) + \frac{1}{m} \sum_{j=1}^{m} \texttt{CrossEntropy}(\hat{\mathbf{d}}_{i^*,j}, \mathbf{m}_{i^*})$
16 $\Phi_G \leftarrow \Phi_G - 0.0002 \times \texttt{Adam}(\nabla_{\Phi_G} \mathcal{L}_G)$
```

Chapter 6

Other Methods for Synthetic Data Generation

Before developing CTGAN, we made several attempts to build a synthetic data generator, including a long-short-term-memory (LSTM) [23] model and a variational autoencoder model. In this chapter, we describe these efforts.

6.1 TGAN

TGAN [55] uses an LSTM to generate synthetic data column by column. Each column depends on the previously generated columns. We use an attention mechanism [5] to model the correlation between columns. When generating a column, the attention mechanism pays attention to previous columns that are highly related to the current column. TGAN has many more parameters than CTGAN, and so is more time-consuming to train.

Preprocessing TGAN uses a similar preprocessing method as CTGAN. All continuous columns are normalized to a scalar $\alpha_{i,j}$ and a vector $\beta_{i,j}$ using mode-specific normalization (Section 5.2).[1] All discrete columns are represented as a one-hot vector $\mathbf{d}_{i,j}$.

[1] In TGAN, we use Gaussian mixture model (GMM) with 5 modes. In CTGAN, GMM is upgraded to VGM to automatically infer the number of modes.

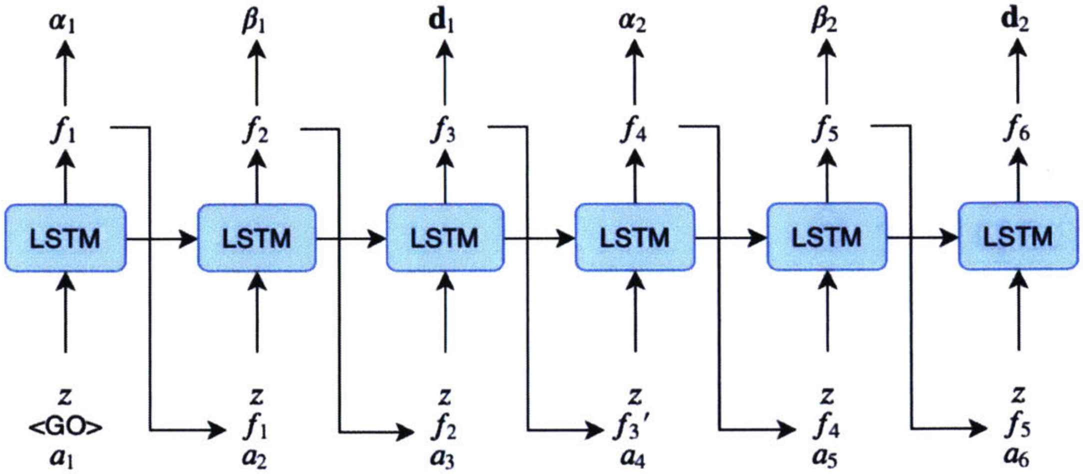


Figure 6-1: Example of using TGAN generator to generate a simple table. The example has two continuous variables and two discrete variables. The order of these columns is $[C_1, D_1, C_2, D_2]$. TGAN generates these four variables one by one following their original order in the table. Each sample is generated in six steps. Each numerical variable is generated in two steps while each categorical variable is generated in one step.

Figure 6-1 shows the structure of our TGAN and how to use it to generate tabular data. We use a long-short-term memory (LSTM) network as the generator, and use Multi-Layer Perceptron (MLP) in the discriminator.

Generator: We generate a numerical variable in two steps. We first generate the value scalar α_i, then generate the cluster vector β_i. We generate a categorical feature in one step as a probability distribution over all possible labels.

The input to the LSTM in each step t is the random variable z, the previous hidden vector f_{t-1} or an embedding vector f'_{t-1} depending on the type of previous output, and the weighted context vector a_t. The random variable z is a 100-dimensional vector sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The attention-based context vector a_t is a weighted average over all the previous LSTM outputs $h_{1:t}$. We learn an attention weight vector $\gamma_t \in \mathbb{R}^{t-1}$. γ_t are additional parameters in the model. The context vector is computed as

$$a_t = \sum_{k=1}^{t-1} \mathtt{softmax}(\gamma_t)_k h_k. \tag{6.1}$$

We set $a_1 = \mathbf{0}$, because there are no previously generated columns. The output of

LSTM h_t is a 100-dimensional vector. We project the output to a hidden vector $f_t = \texttt{FC}_{100\to100}(h_t)$. We further convert the hidden vector to an output variable.

- If the output is the value part of a continuous variable, we compute the output as $\alpha_i = \texttt{FC}_{100\to1}(f_t)$. The hidden vector for $t+1$ step is f_t.

- If the output is the cluster membership of a continuous variable, we compute the output as $\beta_i = \texttt{softmax}(\texttt{FC}_{100\to5}(f_t))$. The feature vector for $t+1$ step is f_t.

- If the output is a discrete variable, we compute the output as $\mathbf{d}_i = \texttt{softmax}(\texttt{FC}_{100\to|D_i|}(f_t))$. The hidden vector for $t+1$ step is $f'_t = E_i[\arg_k \max \mathbf{d}_i]$, where $E \in \mathbb{R}^{|D_i|\times100}$ is an embedding matrix for discrete column D_i.

- f_0 is a special vector <GO> and we learn it during the training.

Discriminator We use a two-layer, fully connected neural network as the discriminator. The network structure is the same as CTGAN. Instead of PacGAN, we use a mini-batch discrimination vector [45] in TGAN.

Loss Function The model is differentiable, so we train our model using an Adam optimizer. We optimize the generator so that it can fool the discriminator as much as possible. To warm up the model more efficiently, we jointly optimize the KL divergence of discrete variables and the cluster vector of continuous variables by adding them to the loss function. Adding the KL divergence term can also make the model more stable. We optimize the generator as

$$\mathcal{L}_\mathcal{G} = -\mathbb{E}_{z\sim\mathcal{N}(\mathbf{0},\mathbf{I})} \log \mathcal{D}(\mathcal{G}(z)) + \sum_{i=1}^{N_c} \mathbb{KL}(\beta'_i, \beta_i) + \sum_{i=1}^{N_d} \mathbb{KL}(\mathbf{d}'_i, \mathbf{d}_i), \tag{6.2}$$

where β'_i and $\mathbf{d}'_i$ are generated data, while β_i and $\mathbf{d}_i$ are real data. We optimize the discriminator in the same way as the vanilla GAN [17].

6.2 Tabular VAE

The VAE simultaneously trains a generative model $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$ and an inference model $q_\phi(\mathbf{z}|\mathbf{x})$ by minimizing the evidence lower-bound (ELBO) loss [31]

$$\log p_\theta(\mathbf{x}_j) \geq \mathcal{L}(\theta, \phi; \mathbf{x}_j) = \mathbb{E}_{q_\phi(z_j|\mathbf{x}_j)}\big[\log p_\theta(\mathbf{x}_j|z_j)] - \mathbb{KL}[q_\phi(z_j|\mathbf{x}_j)||p(z_j)]. \qquad (6.3)$$

where

$$\log p_\theta(\mathbf{x}) = \log p_\theta(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = \sum_{j=1}^{n} \log p_\theta(\mathbf{x}_j)$$

Usually $p(z_j)$ is multivariate Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Moreover, $p_\theta(\mathbf{x_j}|\mathbf{z_j})$ and $q_\phi(\mathbf{z_j}|\mathbf{x_j})$ are parameterized using neural networks and optimized using gradient descent.

When using VAE to model rows $\mathbf{r}_j$ in tabular data $\mathbf{T}$, each row is preprocessed as

$$\mathbf{r}_j = \texttt{cat}(\alpha_{1,j}, \beta_{1,j}, \ldots, \alpha_{N_c,j}, \beta_{N_c,j}, \mathbf{d}_{1,j}, \ldots, \mathbf{d}_{N_d,j}).$$

This affects the design of the network $p_\theta(\mathbf{r}_j|z_j)$ which needs to be done differently so that $p_\theta(\mathbf{r}_j|z_j)$ can be modeled accurately and trained effectively. In our design, the neural network outputs a joint distribution of $2N_c + N_d$ variables, corresponding to $2N_c + N_d$ variables $\mathbf{r}_j$. We assume $\alpha_{i,j}$ follows a Gaussian distribution with different means and variance. All $\beta_{i,j}$ and $\mathbf{d}_{i,j}$ follow a categorical PMF. Here is our design.

$$\begin{cases} h_1 = \texttt{ReLU}(\texttt{FC}_{128\to128}(z_j)) & \\ h_2 = \texttt{ReLU}(\texttt{FC}_{128\to128}(h_1)) & \\ \bar{\alpha}_{i,j} = \texttt{tanh}(\texttt{FC}_{128\to1}(h_2)) & 1 \leq i \leq N_c \\ \hat{\alpha}_{i,j} \sim \mathcal{N}(\bar{\alpha}_{i,j}, \delta_i) & 1 \leq i \leq N_c \\ \hat{\beta}_{i,j} \sim \text{Cat}(\texttt{softmax}(\texttt{FC}_{128\to m_i}(h_2))) & 1 \leq i \leq N_c \\ \hat{\mathbf{d}}_{i,j} \sim \text{Cat}(\texttt{softmax}(\texttt{FC}_{128\to|D_i|}(h_2))) & 1 \leq i \leq N_d \\ p_\theta(\mathbf{r}_j|z_j) = \prod_{i=1}^{N_c} \mathbb{P}(\hat{\alpha}_{i,j} = \alpha_{i,j}) \prod_{i=1}^{N_c} \mathbb{P}(\hat{\beta}_{i,j} = \beta_{i,j}) \prod_{i=1}^{N_d} \mathbb{P}(\hat{\alpha}_{i,j} = \alpha_{i,j}) & \end{cases}$$

Here $\hat{\alpha}_{i,j}, \hat{\beta}_{i,j}, \hat{\mathbf{d}}_{i,j}$ are random variables, and $p_\theta(\mathbf{r}_j|z_j)$ is the joint distribution of these variables. So $\log p_\theta(\mathbf{r}_j|z_j)$ is

$$\sum_{i=1}^{N_c} \log \frac{1}{\sqrt{2\pi\delta_i}} \exp \frac{(\alpha_{i,j} - \bar{\alpha}_{i,j})}{2\delta_i^2} + \sum_{i=1}^{N_c} \mathbb{CE}(\hat{\beta}_{i,j}, \beta_{i,j}) + \sum_{i=1}^{N_d} \mathbb{CE}(\hat{\mathbf{d}}_{i,j}, \mathbf{d}_{i,j}) + \text{constant}. \quad (6.4)$$

In $p_\theta(\mathbf{r}_j|z_j)$, weight matrices and δ_i are parameters in the network. These parameters are trained using gradient descent.

The modeling for $q_\phi(z_j|\mathbf{r}_j)$ is similar to conventional VAE.

$$\begin{cases}
\mathbf{r}_j = \texttt{cat}(\alpha_{1,j}, \beta_{1,j}, \ldots, \alpha_{N_c,j}, \beta_{N_c,j}, \mathbf{d}_{1,j}, \ldots, \mathbf{d}_{N_d,j}) \\
h_1 = \texttt{ReLU}(\texttt{FC}_{|\mathbf{r}_j| \to 128}(\mathbf{r}_j)) \\
h_2 = \texttt{ReLU}(\texttt{FC}_{128 \to 128}(h_1)) \\
\mu = \texttt{FC}_{128 \to 128}(h_2) \\
\sigma = \exp(\frac{1}{2}\texttt{FC}_{128 \to 128}(h_2)) \\
q_\phi(z_j|\mathbf{r}_j) \sim \mathcal{N}(\mu, \sigma\mathbf{I})
\end{cases}$$

`TVAE` is trained using Adam with learning rate 1e-3.

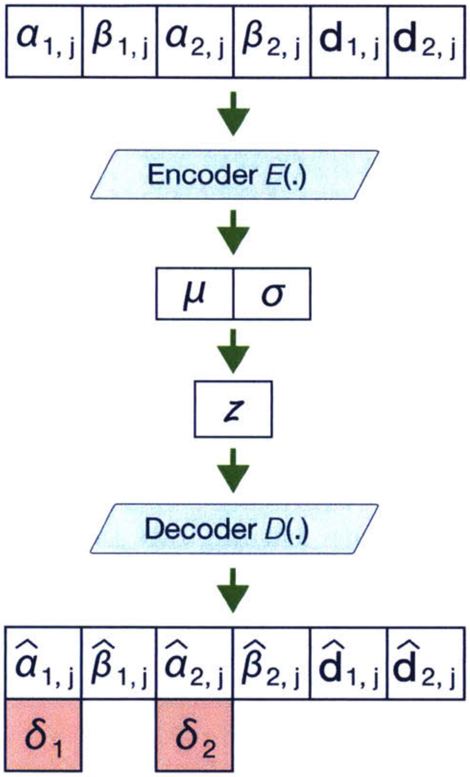


Figure 6-2: TVAE structure.

Chapter 7

SDGym Benchmark Framework

Many statistical and deep learning methods have been proposed to model tabular data. However, as of this writing, no consistent benchmark has been developed to fairly compare different methods. Tabular data have different properties, such as single or mixed column types, low or high dimensions, etc. A thorough benchmark should contain various combinations of such properties.

As shown in [53], metrics for evaluating generative models are largely independent. Thus, we don't rely on one metric to compare different models. Instead, our benchmark contains two parts, simulated data and real data, and evaluates several metrics. Simulated data have a known probability distribution and are used to evaluate the quality of learned data distribution, whereas real data come from a real machine learning task and can be used to evaluate performance in a real scenario.

In this benchmark, we focus on how well a model can learn the probability distribution of rows. The benchmark does not challenge models for learning distributions from very few examples. All the datasets we select contain at least $10,000$ rows, and the number of rows is significantly higher than the number of columns.

In the rest of this chapter, we describe the simulated data and real data benchmark aspects in detail.

7.1 Simulated data

For the simulated data, we handcrafted several data distributions, using a Gaussian mixture model or a Bayesian network in order to control the data distribution. As shown in Figure 7-1, we handcrafted a simulated data generator $\mathcal{S}$ and generated training and test set $\mathbf{T}_{train}$ and $\mathbf{T}_{test}$. We trained a data synthesizer G on $\mathbf{T}_{train}$ and generated synthetic data $\mathbf{T}_{syn}$. We evaluated G using the following metrics.

- **Likelihood of generated data $\mathcal{L}_{syn}$**: We computed the likelihood of $\mathbf{T}_{syn}$ on $\mathcal{S}$. Because $\mathcal{S}$ is a known distribution, the likelihood of synthetic data on the simulated data distribution can be easily computed. The likelihood can show, in some sense, the quality of synthetic data. This is a flawed metric: High likelihood does not necessarily indicate a good synthesizer, as a trivial G which repeats the same data point multiple times could achieve high likelihood.

- **Likelihood of test data $\mathcal{L}_{test}$**: We retrain the simulated data generator as $\mathcal{S}'$ using $\mathbf{T}_{syn}$. $\mathcal{S}'$ has the same structure as $\mathcal{S}$, but different parameters. If $\mathcal{S}$ is a Gaussian mixture model, we use the same number of Gaussian components and retrain the mean and covariance of each component. If $\mathcal{S}$ is a Bayesian network, we keep the same graph and learn a new conditional distribution on each edge. We compute the likelihood of $\mathbf{T}_{test}$ on $\mathcal{S}'$. This metric introduces the prior knowledge of the structure of $\mathcal{S}'$ which is not necessarily encoded in $\mathbf{T}_{syn}$, thus may be a flawed metric.

We constructed 7 different simulated data sets. Table 7.1 shows statistical information about the simulated data. `Grid`, `gridr` and `ring` are generated by Gaussian mixture models. We follow [50] to generate `grid` and `ring`. `Gridr` is generated by adding random noise to the modes in `grid`. Figure B-1 shows these three datasets. The other four datasets are generated by Bayesian networks. Figure B-2 and B-3 shows the Bayesian network structures. We use graph structures and probability distributions from `http://www.bnlearn.com/bnrepository/`.

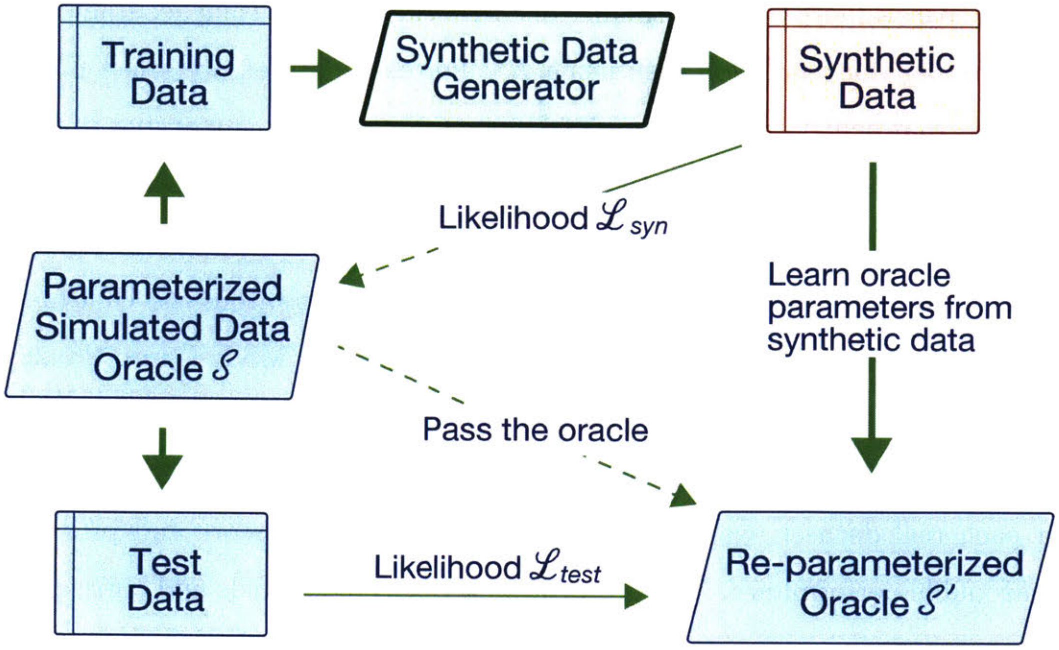

Training Data
Synthetic Data Generator
Synthetic Data
Likelihood $\mathscr{L}_{syn}$
Parameterized Simulated Data Oracle $\mathcal{S}$
Learn oracle parameters from synthetic data
Pass the oracle
Test Data
Likelihood $\mathscr{L}_{test}$
Re-parameterized Oracle $\mathcal{S}'$

Figure 7-1: Evaluate synthetic data generator using simulated data.

7.2 Real data

For the real data component, we wanted to evaluate the effectiveness of using synthetic data as training data for machine learning. In other words, real data enables an application-level evaluation. As shown on Figure 7-2, we have training data $\mathbf{T}_{train}$ and test data $\mathbf{T}_{test}$. We train a data synthesizer G on $\mathbf{T}_{train}$ and generate synthetic data $\mathbf{T}_{syn}$ using G. We train prediction models on $\mathbf{T}_{syn}$ and test prediction models

Table 7.1: Simulated datasets in our benchmark. #C, #B, and #M mean number of continuous columns, binary columns and multi-class discrete columns respectively.

name	#train/test	#C	#B	#M
grid	10k/10k	2	0	0
gridr	10k/10k	2	0	0
ring	10k/10k	2	0	0
asia	10k/10k	0	8	0
alarm	10k/10k	0	13	24
child	10k/10k	0	8	12
insurance	10k/10k	0	8	19

using $\mathbf{T}_{test}$. We evaluate the performance of classification tasks using accuracy and F1, while evaluating the regression task using R^2. For each dataset, we select classifiers or regressors that achieve reasonable performance. Since we are not trying to pick the best classification or regression model, we take the average performance of multiple prediction models as metrics for G.

We pick `adult`, `census`, `covertype`, `intrusion` and `news` from the UCI machine learning repo. We pick `credit` from Kaggle. We convert MNIST into 28x28 and 12x12 images, vectorize each image as one row of data, and call them `MNIST28` and `MNIST12` respectively. Table 7.2 shows statistics for these simulated datasets.

For each real dataset, we run several classifiers and regressors and pick models with reasonable performance. Table 7.3 shows the selected models and corresponding performances.

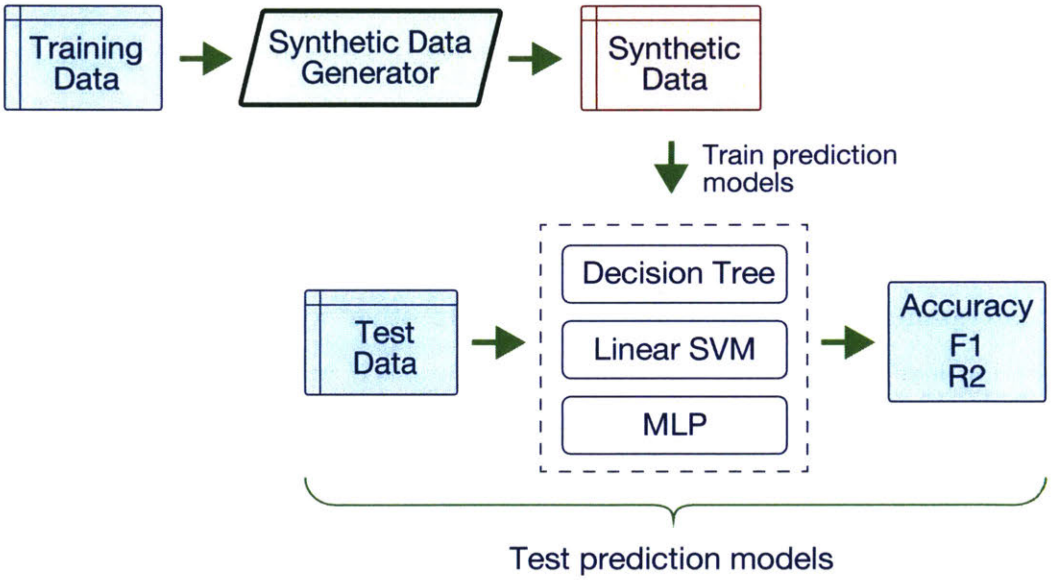


Figure 7-2: Real data in synthetic data generator benchmark.

Table 7.2: Real datasets in our benchmark. #C, #B, and #M mean number of continuous columns, binary columns and multi-class discrete columns respectively.

name	#train	#test	#col	#C	#B	#M	task
adult	22561	10000	15	6	2	7	classification
census	199523	99762	41	7	3	31	classification
covtype	481012	100000	55	10	44	1	classification
credit	264807	20000	30	29	1	0	classification
intrusion	394021	100000	41	26	5	10	classification
mnist12	60000	10000	145	0	144	1	classification
mnist28	60000	10000	785	0	784	1	classification
news	31644	8000	59	45	14	0	regression

Table 7.3: Classifiers and regressors selected for each real dataset and corresponding performance.

dataset	name	accuracy	f1	macro_f1	micro_f1	r2
adult	Adaboost (estimator=50)	86.07%	68.03%			
	Decision Tree (depth=20)	79.84%	65.77%			
	Logistic Regression	79.53%	66.06%			
	MLP (50)	85.06%	67.57%			
census	Adaboost (estimator=50)	95.22%	50.75%			
	Decision Tree (depth=30)	90.57%	44.97%			
	MLP (100)	94.30%	52.43%			
covtype	Decision Tree (depth=30)	82.25%		73.62%	82.25%	
	MLP (100)	70.06%		56.78%	70.06%	
credit	Adaboost (estimator=50)	99.93%	76.00%			
	Decision Tree (depth=30)	99.89%	66.67%			
	MLP (100)	99.92%	73.31%			
intrusion	Decision Tree (depth=30)	99.91%		85.82%	99.91%	
	MLP (100)	99.93%		86.65%	99.93%	
mnist12	Decision Tree (depth=30)	84.10%		83.88%	84.10%	
	Logistic Regression	87.29%		87.11%	87.29%	
	MLP (100)	94.40%		94.34%	94.40%	
mnist28	Decision Tree (depth=30)	86.08%		85.89%	86.08%	
	Logistic Regression	91.42%		91.29%	91.42%	
	MLP (100)	97.28%		97.26%	97.28%	
news	Linear Regression					0.1390
	MLP (100)					0.1492

Chapter 8

Experiment Results

In this chapter, we use SDGym to benchmark existing statistical and deep learning models, as well as CTGAN and TVAE. We first describe our experimental settings and hyperparameters. We then show our quantitative results, go through some case studies, and present an ablation study on CTGAN.

8.1 Settings and hyperparameters

We evaluate CLBN, PrivBN, MedGAN, VeeGAN, TableGAN, CTGAN, and TVAE using our SDGym. Here are the settings for each model:

- CLBN: We use the implementation in Pomegranate library [48]. Since CLBN only supports discrete variables, we discretize continuous columns to 15 bins before training the Bayesian network.

- PrivBN: We use the original C++ implementation[1] and write a Python wrapper for the executable binary file. In the implementation, all continuous variables are discretized to 20 bins. PrivBN is a differentially private data synthesizer. It adds noise to synthetic data. For a fair comparison, we use a large privacy budget to reduce the effect of noise on the performance. We set ϵ-differential privacy budget to 10.

[1] https://sourceforge.net/projects/privbayes

- MedGAN: We follow the settings in the original implementation[2]. We use a 1-layer multilayer perceptron (MLP) with 128 hidden units for the autoencoder. And we use 2-layer MLP for GAN. The hidden layer size is $(128, 128)$ for the generator and $(256, 128)$ for the discriminator. We use a relatively large regularization weight equal to $1e-3$.

- VeeGAN: We use 2-layer MLP with hidden size $(128, 128)$ for both generator and reconstructor. We use a 1-layer MLP with hidden size 128 for the discriminator.

- TableGAN: We use 3 convolutional layers in the generator and 3 deconvolutional layers in the discriminator.

- TVAE and CTGAN: We obey the hyperparameters described in Chapter 5.

For TVAE and all GAN-based models, we use batch size 500. Each model is trained for 300 epochs using an Adam optimizer. Each epoch iterates over all the training examples one time.

We posit that for any dataset, across any metrics except $\mathcal{L}_{syn}$, the best performance is achieved by $\mathbf{T}_{train}$. Thus we present the Identity method which outputs $\mathbf{T}_{train}$.

8.2 Quantitative results

Experimental results are shown in Table 8.1, 8.2 and 8.3. The number in the bracket is the rank of a method (lower is better). The rank is computed as follows. For each set of experiments: (1) Rank the algorithms' overall metrics in each set. (2) Take the average of all ranks of each algorithm. Get one score in the range $[1, 7]$ for each algorithm. (3) Rank the scores again.

In the continuous data case, CLBN and PrivBN suffer because continuous data are discretized. MedGAN, VeeGAN, and TableGAN all suffer from mode collapse. With mode-specific normalization, our model performs well on 2D continuous datasets.

[2] https://github.com/mp2893/medgan

Table 8.1: Benchmark results on Gaussian mixture simulated data.

	grid		gridr		ring	
method	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$
Identity	-3.06	-3.06	-3.06	-3.07	-1.70	-1.70
CLBN(2)	-3.68	-8.62	-3.76	-11.60	-1.75	**-1.70**
PrivBN(4)	-4.33	-21.67	-3.98	-13.88	-1.82	-1.71
MedGAN(7)	-10.04	-62.93	-9.45	-72.00	-2.32	-45.16
VEEGAN(6)	-9.81	-4.79	-12.51	-4.94	-7.85	-2.92
TableGAN(5)	-8.70	-4.99	-9.64	-4.70	-6.38	-2.66
TVAE(1)	**-2.86**	-11.26	**-3.41**	**-3.20**	**-1.68**	-1.79
CTGAN(3)	-5.63	**-3.69**	-8.11	-4.31	-3.43	-2.19

Table 8.2: Benchmark results on Bayesian network simulated data.

	asia		alarm		child		insurance	
method	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$	$\mathcal{L}_{syn}$	$\mathcal{L}_{test}$
Identity	-2.23	-2.24	-10.3	-10.3	-12.0	-12.0	-12.8	-12.9
CLBN(3)	-2.44	-2.27	-12.4	-11.2	-12.6	-12.3	-15.2	-13.9
PrivBN(1)	**-2.28**	**-2.24**	-11.9	-10.9	-12.3	**-12.2**	-14.7	**-13.6**
MedGAN(5)	-2.81	-2.59	**-10.9**	-14.2	-14.2	-15.4	-16.4	-16.4
VEEGAN(7)	-8.11	-4.63	-17.7	-14.9	-17.6	-17.8	-18.2	-18.1
TableGAN(6)	-3.64	-2.77	-12.7	-11.5	-15.0	-13.3	-16.0	-14.3
TVAE(2)	-2.31	-2.27	-11.2	**-10.7**	**-12.3**	-12.3	**-14.7**	-14.2
CTGAN(4)	-2.56	-2.31	-14.2	-12.6	-13.4	-12.7	-16.5	-14.8

On the dataset generated from Bayesian networks, CLBN and PrivBN have a natural advantage. Our CTGAN achieves slightly better performance than MedGAN and TableGAN. Surprisingly, TableGAN works well on discrete datasets, despite considering discrete columns as continuous values. Our reasoning for this is that in our simulated data, most columns have fewer than 4 categories, so conversion does not cause serious problems.

On real datasets, TVAE and CTGAN outperform CLBN and PrivBN, whereas other GAN models cannot get as good a result as with Bayesian networks. When it comes to large-scale real datasets, learning a high-quality Bayesian network is difficult. There is a significant performance gap between real data and synthetic data generated by a learned Bayesian network.

Table 8.3: Benchmark results on real data.

	adult	census	credit	cover.	intru.	mnist12/28		news
method	F1	F1	F1	Macro	Macro	Acc	Acc	R^2
Identity	0.669	0.494	0.720	0.652	0.862	0.886	0.916	0.14
CLBN(3)	0.334	0.310	0.409	0.319	0.384	0.741	0.176	-6.28
PrivBN(4)	0.414	0.121	0.185	0.270	0.384	0.117	0.081	-4.49
MedGAN(6)	0.375	0.000	0.000	0.093	0.299	0.091	0.104	-8.80
VEEGAN(6)	0.235	0.094	0.000	0.082	0.261	0.194	0.136	-6.5e6
TableGAN(5)	0.492	0.358	0.182	0.000	0.000	0.100	0.000	-3.09
TVAE(1)	**0.626**	0.377	0.098	**0.433**	0.511	**0.793**	**0.794**	**-0.20**
CTGAN(1)	0.601	**0.391**	**0.672**	0.324	**0.528**	0.394	0.371	-0.43

We compute the distance between generated synthetic data and nearest neighbor in training data on all real datasets shown in Table 8.4. TVAE and CTGAN are both at the top in terms of machine learning effectiveness, but synthetic data generated by CTGAN has a larger distance to the training data then TVAE. Thus, CTGAN preserves privacy better than TVAE.

Table 8.4: Distance between synthetic data and nearest neighbor in training set.

Model	Avg.	Std.
CLBN(3)	59.5	145.1
PrivBN(4)	8.6	10.0
MedGAN(6)	1073.5	2763.3
VeeGAN(6)	4.7	2.6
TableGAN(5)	2.5	0.9
TVAE(1)	2.5	2.0
CTGAN(1)	3.1	2.7

On seven classification datasets, we mostly use decision tree classifiers (DCT) and MLP classifiers. Table 8.5 shows the average performance of these two classifiers. On the real training data, MLP can achieve slightly better performance than decision trees, but does not work as well on synthetic data generated by Bayesian-based synthetic data generators. On synthetic data generated by MedGAN, VeeGAN and TableGAN, DCT and MLP achieve similar performance. Our TVAE and CTGAN generate high-quality synthetic data, so both DCT and MLP perform much better than trained on other synthetic data. MLP is slightly better than DCT, which is consistent with real data.

Table 8.5: Classification performance of different classifers.

Model	DCT	MLP
Identity	0.724	0.755
CLBN	0.409	0.350
PrivBN	0.267	0.216
MedGAN	0.110	0.148
VeeGAN	0.148	0.139
TableGAN	0.282	0.288
TVAE	0.477	0.532
CTGAN	0.437	0.509

8.3 Case analysis

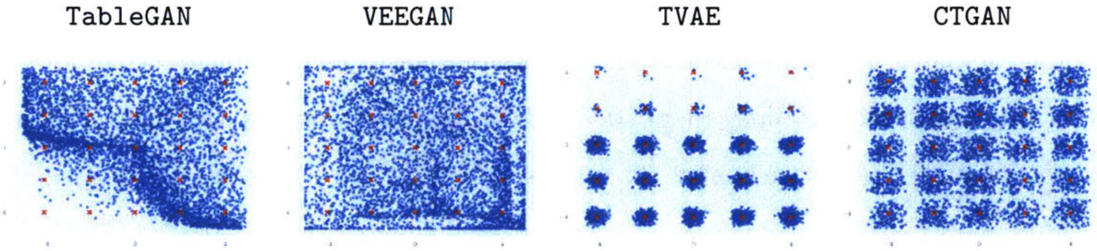


Figure 8-1: Visualize synthesized grid data set using TableGAN, VEEGAN, TVAE and CTGAN. The red marks are the ground truth modes and the blue dots are synthetic samples generated by different synthesizers.

Figure 8-1 shows the results of TableGAN, VEEGAN, TVAE and CTGAN on the grid dataset. Clearly, TableGAN and VEEGAN do not model the data very well. TableGAN ignores the bottom left corner and does not properly model the variance on any of the modes. VeeGAN also fails, as many samples lie at the boundary. VeeGAN also tends to sample lots of data horizontally or vertically between adjacent modes. For example, many points are sampled on the horizontal line $y = -4$ and the vertical line $x = 4$. Our TVAE and CTGAN successfully capture 25 modes, largely because the reversible data transformation can successfully capture five modes on each axis. TVAE captures the variance better than CTGAN. Synthetic data generated by CTGAN has a higher variance than ground truth. Surprisingly, in TVAE, the first two rows get fewer samples than the last three rows.

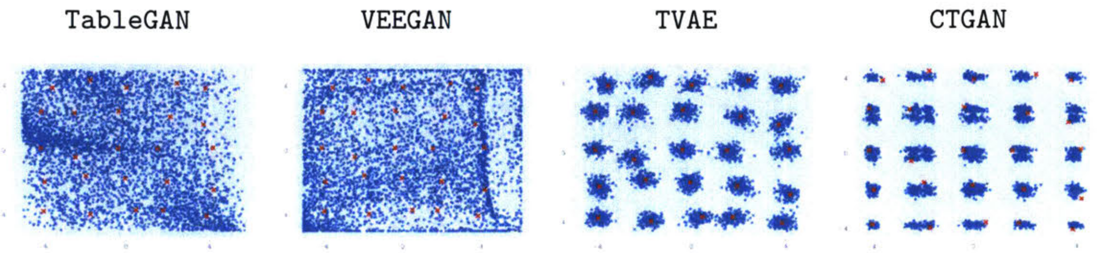


Figure 8-2: Visualize synthesized gridr data set using TVAE and CTGAN. The red marks are the ground truth modes and the blue dots are synthetic samples generated by different synthesizers.

Figure 8-2 shows TVAE and CTGAN results on the gridr dataset. In this dataset, the modes are randomly perturbed, so that the modes found in the Gaussian mixture model preprocessing are not perfectly aligned with the modes in the data. Therefore, the generative model needs to properly generate one dimension of data depending on the value of the other dimension. TVAE works perfectly on this dataset. It does not suffer from the same problem in grid. CTGAN does not work as well as TVAE, as it does not capture the perturbation on modes.

8.4 Ablation study

To understand the importance of each module in CTGAN, we conducted an ablation study.

Mode-specific normalization: In CTGAN, we use a variational Gaussian mixture model (VGM) to normalize continuous columns. We compare it with (1) GMM5: Gaussian mixture model with 5 modes, (2) GMM10: Gaussian mixture model with 10 modes, and (3) MinMax: min-max normalization to $[-1, 1]$. Table 8.6 shows that using GMM slightly decreases the performance while min-max normalization gives the worst performance.

Conditional vector and training-by-sampling: We successively remove these two

Table 8.6: Ablation study on mode-specific normalization.

Model	GMM5	GMM10	MinMax
Performance	-4.1%	-8.6%	-25.7%

components. (1) `w/o S.`: we first disable the training-by-sampling, but the generator still gets a condition vector and its loss function still has the cross-entropy term. The condition vector is sampled from training data frequency instead of log frequency. (2) `w/o C.`: We further remove the condition vector in the generator. Table 8.7 shows that both training-by-sampling and the condition vector are important for imbalanced datasets. Especially on highly imbalanced datasets such as `credit`, removing training-by-sampling results in 0% on the F1 metric.

Table 8.7: Ablation study on training-by-sampling and conditional vector.

Model	w/o S.	w/o C.
Performance	-17.8%	-36.5%

WGANGP and PacGAN: In the thesis, we use WGANGP+PacGAN. Here we compare it with three alternatives: WGANGP only, vanilla GAN loss only, and vanilla GAN + PacGAN. Table 8.8 shows that WGANGP is more suitable for a synthetic data task than vanilla GAN, while PacGAN is helpful for vanilla GAN loss but not as important for WGANGP.

Table 8.8: Ablation study on Wasserstein GAN and PacGAN.

Model	GAN	WGANGP	GAN+PacGAN
Performance	-6.5%	+1.75%	-5.2%

8.5 Discussion

`TVAE` outperforms `CTGAN` in several cases, but GANs do have several favorable attributes, and this result does not indicate that we should always use VAEs rather than GANs on modeling tables. `CTGAN` has a few advantages over `TVAE`, namely

- since the generator in GAN is not directly optimized by mean square error, it is easier to make it differentially private using existing frameworks like DPGAN and PATE-GAN. Empirically, we compute the distance between synthetic data and nearest neighbor in training data. We observe that `CTGAN` gets a 13% larger distance than `TVAE`, while achieving the same accuracy or F1 score on the real data.

- `CTGAN` is more flexible in the sense that it is capable of capturing interactions among variables through their architecture, while `TVAE` is not intrinsically capable of doing so. To this end, in scenarios where strong complex underlying structures are involved, `CTGAN` should outperform `TVAE`.

Chapter 9

Conclusion and Future Work

In this thesis, a comprehensive synthetic tabular data benchmark, SDGym, was implemented and open-sourced. The implementation of the benchmark revealed the challenges of modeling tabular data and problems in existing statistical and deep learning models. CTGAN was designed to overcome such problems. Experimental results show that CTGAN can model tabular data with complicated distributions and mixed types, and is robust over different datasets. CTGAN is the first deep learning model that outperforms Bayesian networks in our benchmark.

There are a number of use cases for synthetic data. CTGAN can effectively model tabular data and generate high-quality synthetic data, but further research is required to satisfy different types of applications. As future work, we would like to explore the following directions.

- **Generate differentially private synthetic data.** One important application involves using synthetic data to overcome privacy or bureaucratic barriers in releasing data. Since CTGAN outperforms all other GAN-based models, it is important to create a differentially private version of CTGAN. Similarly, We want to explore the possibility of differentially private VAEs.

- **Apply CTGAN on data augmentation.** A lack of training data is a severe problem in machine learning. Synthetic data can be used to generate a large volume of training data. We want to explore whether machine learning models can benefit from adding extra synthetic training data into a real training set.

- **Use `CTGAN` to understand underlying relations between columns.** Understanding the causal relationship between columns is challenging. We want to interpret the learned generative model in order to understand the underlying relation between columns.

- **Understand the theoretical guarantee of model convergence.** Although we show empirically that GANs can be used to model a mix of discrete and continuous variables, a theoretical justification is required to guarantee that the model works as expected. Such a guarantee could also inspire better learning objectives and optimization methods for gaining better performance.

Appendix A

Notations

Table A.1: Notations

Notation	Description
	Notations for probability distributions.
$\mathcal{N}(\mu, \sigma\mathbf{I})$	Multivariate Gaussian distribution.
$\mathcal{U}(l, h)$	Uniform distribution on $[l, r]$.
	Notations for tabular data generation task.
$C_1, \dots, C_{N_c}$	N_c continuous columns in tabular data.
$D_1, \dots, D_{N_d}$	N_d discrete columns in tabular data.
$\mathbf{T}$	Real or simulated tabular data with $N_c + N_d$ columns.
$\mathbf{T}_{train}$, $\mathbf{T}_{test}$	The traning and testing part of $\mathbf{T}$.
$\mathbf{T}_{syn}$	Synthetic data generated by some generative model.
$c_{i,j}$	A float showing the i-th continuous column of j-th row in $\mathbf{T}_{train}$.
$d_{i,j}$	A integer showing the i-th discrete column of j-th row in $\mathbf{T}_{train}$.
	Notations for the benchmark.
$\mathcal{S}, \mathcal{S}'$	Simulated data generator and the retrained generator in the benchmark.
G	A synthetic data generation model to be evaluated.
	Notations for preprocessing.
$\alpha_{i,j}$	A normalized value for $c_{i,j}$ row.
$\beta_{i,j}$	A one-hot vector denoting the mode $c_{i,j}$ coming from m_i Gaussian distributions.
$\mathbf{d}_{i,j}$	A one-hot representation for $d_{i,j}$.
	Notations for `CTGAN`.
$\mathcal{G}(\cdot)$	Generator.
$\mathcal{C}(\cdot)$	Critic.
z	Random noise input.
	Notations for `TVAE`.
$\mathcal{E}(\cdot)$	Encoder.
$\mathcal{D}(\cdot)$	Decoder.
μ, σ	Mean and standard deviation of the hidden representation.
z	Hidden representation.

Appendix B

Figures

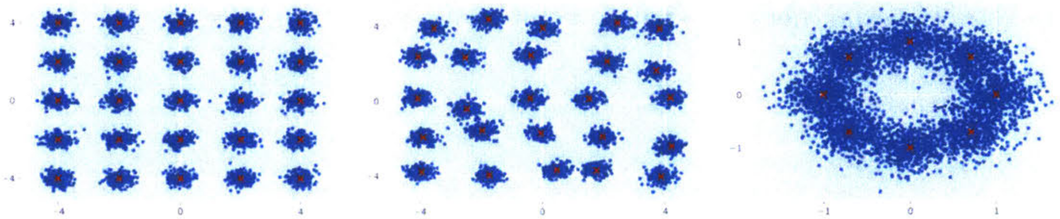


Figure B-1: Visualize `grid` (left), `gridr` (middle) and `ring` (right) datasets. Each blue point represents a row in a table. The x-axis and y-axis represent the value of two continuous columns respectively. The red ‘x’s in the plots are the modes for Gaussian mixtures.

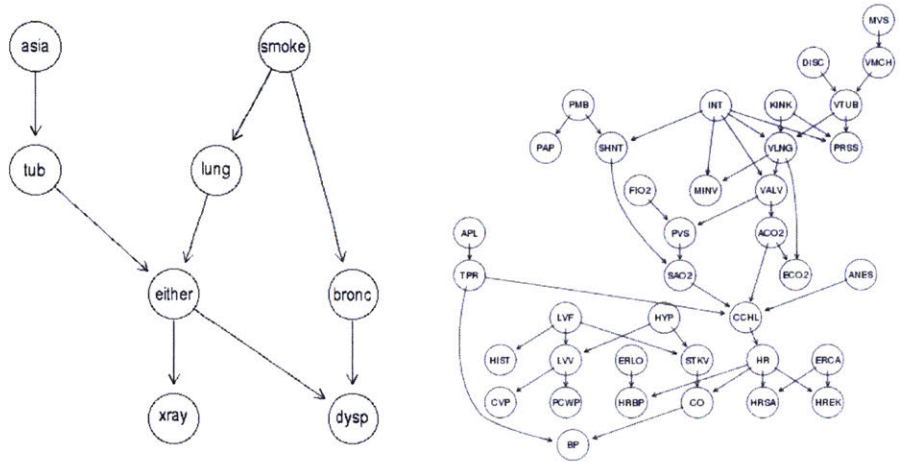


Figure B-2: Bayesian network structures of asia (left) and alarm (right) datasets.

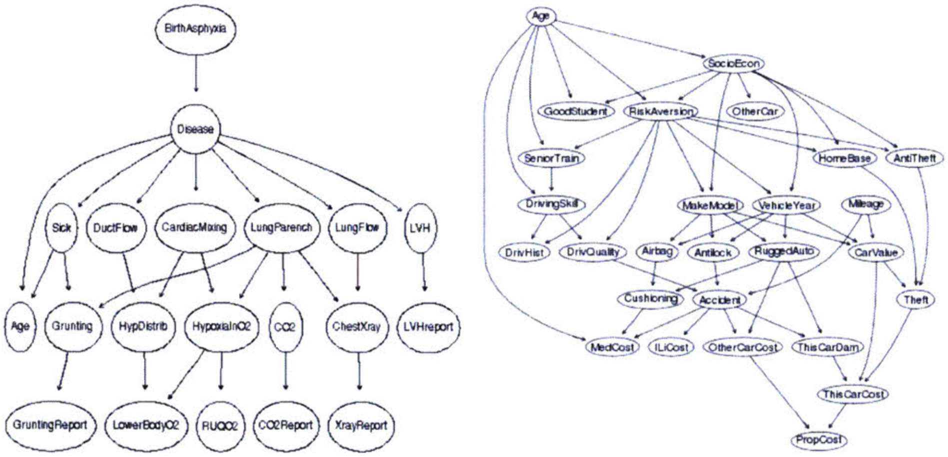


Figure B-3: Bayesian network structures of child (left) and insurance (right) datasets.

Bibliography

[1] The state of ml and data science 2017.

[2] Charu C Aggarwal et al. *Recommender systems*, volume 1. Springer, 2016.

[3] Adel Alaeddini, Kai Yang, Chandan Reddy, and Susan Yu. A probabilistic model for predicting the probability of no-show in hospital appointments. *Health care management science*, 14(2):146–157, 2011.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, 2017.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[6] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. Citeseer, 2007.

[7] Christopher M Bishop. *Pattern recognition and machine learning.* springer, 2006.

[8] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[9] Ramiro Camino, Christian Hammerschmidt, and Radu State. Generating multi-categorical samples with generative adversarial networks. In *ICML workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

[10] George Casella and Roger L Berger. *Statistical inference*, volume 2. Duxbury Pacific Grove, CA, 2002.

[11] Zhengping Che, Yu Cheng, Shuangfei Zhai, Zhaonan Sun, and Yan Liu. Boosting deep learning risk prediction with generative adversarial networks for electronic health records. In *International Conference on Data Mining.* IEEE, 2017.

[12] Irene Chen, Fredrik D Johansson, and David Sontag. Why is my classifier discriminatory? In *Advances in Neural Information Processing Systems*, pages 3539–3550, 2018.

[13] Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F. Stewart, and Jimeng Sun. Generating multi-label discrete patient records using generative adversarial networks. In *Machine Learning for Healthcare Conference*. PMLR, 2017.

[14] C Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

[15] Cynthia Dwork. Differential privacy. *Encyclopedia of Cryptography and Security*, pages 338–340, 2011.

[16] Maayan Frid-Adar, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. Synthetic data augmentation using gan for improved liver lesion classification. In *2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018)*, pages 289–293. IEEE, 2018.

[17] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

[18] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, 2017.

[19] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[20] Andrew J. Hawkins. Uber settles claims that it mishandled private information about users and drivers, Aug 2017.

[21] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[22] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[23] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[24] Zan Huang, Daniel Zeng, and Hsinchun Chen. A comparison of collaborative-filtering recommendation algorithms for e-commerce. *IEEE Intelligent Systems*, 22(5):68–78, 2007.

[25] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on International Conference on Machine Learning*, 2015.

[26] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[27] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2016.

[28] Bargav Jayaraman, Lingxiao Wang, David Evans, and Quanquan Gu. Distributed learning without distress: Privacy-preserving empirical risk minimization. In *Advances in Neural Information Processing Systems*, pages 6343–6354, 2018.

[29] Noah Johnson, Joseph P Near, and Dawn Song. Towards practical differential privacy for sql queries. *Proceedings of the VLDB Endowment*, 11(5):526–539, 2018.

[30] James Jordon, Jinsung Yoon, and Mihaela van der Schaar. Pate-gan: Generating synthetic data with differential privacy guarantees. In *International Conference on Learning Representations*, 2019.

[31] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2013.

[32] Satkartar K Kinney, Jerome P Reiter, Arnold P Reznek, Javier Miranda, Ron S Jarmin, and John M Abowd. Towards unrestricted public use business microdata: The synthetic longitudinal business database. *International Statistical Review*, 79(3):362–384, 2011.

[33] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[34] Joonseok Lee, Seungyeon Kim, Guy Lebanon, Yoram Singer, and Samy Bengio. Llorma: local low-rank matrix approximation. *The Journal of Machine Learning Research*, 2016.

[35] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David M Blei. Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence. In *Proceedings of the 10th ACM conference on recommender systems*, pages 59–66, 2016.

[36] Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. Pacgan: The power of two samples in generative adversarial networks. In *Advances in Neural Information Processing Systems*, 2018.

[37] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[38] Mark O'Connor and Jon Herlocker. Clustering items for collaborative filtering. In *SIGIR workshop on recommender systems*. UC Berkeley, 1999.

[39] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. Data synthesis based on generative adversarial networks. In *International Conference on Very Large Data Bases*, 2018.

[40] Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. The synthetic data vault. In *International Conference on Data Science and Advanced Analytics*. IEEE, 2016.

[41] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations*, 2015.

[42] Trivellore E Raghunathan, Jerome P Reiter, and Donald B Rubin. Multiple imputation for statistical disclosure limitation. *Journal of Official Statistics*, 19(1):1, 2003.

[43] Jerome P Reiter. Simultaneous use of multiple imputation for missing data and disclosure limitation. *Journal of Survey Methodology*, 30(2):235–242, 2004.

[44] Jerome P Reiter. Releasing multiply imputed, synthetic public use microdata: An illustration and empirical study. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 168(1):185–205, 2005.

[45] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[46] Aarti Sathyanarayana, Shafiq Joty, Luis Fernandez-Luque, Ferda Ofli, Jaideep Srivastava, Ahmed Elmagarmid, Teresa Arora, and Shahrad Taheri. Sleep quality prediction from wearable data using deep learning. *JMIR mHealth and uHealth*, 4(4):e125, 2016.

[47] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

[48] Jacob Schreiber. Pomegranate: fast and flexible probabilistic modeling in python. *The Journal of Machine Learning Research*, 18(1):5992–5997, 2017.

[49] Ryan Singel. Netflix spilled your brokeback mountain secret, lawsuit claims, Jan 2018.

[50] Akash Srivastava, Lazar Valkov, Chris Russell, Michael U Gutmann, and Charles Sutton. Veegan: Reducing mode collapse in gans using implicit variational learning. In *Advances in Neural Information Processing Systems*, 2017.

[51] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[52] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1441–1450, 2019.

[53] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations*, 2016.

[54] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 2017.

[55] Lei Xu and Kalyan Veeramachaneni. Synthesizing tabular data using generative adversarial networks. *arXiv preprint arXiv:1811.11264*, 2018.

[56] Alexandre Yahi, Rami Vanguri, Noémie Elhadad, and Nicholas P Tatonetti. Generative adversarial networks for electronic health records: A framework for exploring and evaluating methods for predicting drug-induced laboratory test trajectories. In *NIPS workshop on machine learning for health care*, 2017.

[57] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI Conference on Artificial Intelligence*, 2017.

[58] Jun Zhang, Graham Cormode, Cecilia M Procopiuc, Divesh Srivastava, and Xiaokui Xiao. Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems*, 42(4):25, 2017.

[59] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *international conference on computer vision*, pages 2223–2232. IEEE, 2017.