Compact Latent Representations via Autoencoders for Sparse Input Data in Synthetic Tabular Generation

Jimmy Woo
MScAC
University of Toronto
jimmywoo@cs.toronto.edu

Sumant Bagri
MScAC
University of Toronto
sbagri@cs.toronto.edu

Vignesh Edithal
MScAC
University of Toronto
edithal@cs.toronto.edu

Abstract

Synthesizing tabular data involves learning distributions for both categorical and continuous variables. Training datasets for such models may include under-represented categorical features as well as multi-modal continuous variables, and devising a singular learning procedure for such a problem is a non-trivial task. CT-GAN and its variants use a conditional generative adversarial network that has been shown to outperform several Bayesian network baselines. However, the quality of the synthetic data is sensitive to sparsity in the input data. In this project, we utilize autoencoders to learn compact latent representations of the input data with the goal of improving the quality of the synthetic data. We evaluate the quality of synthetic data generated using a vanilla autoencoder, denoising autoencoder, variational autoencoder and entity embedding autoencoder across 5 different datasets and compare them with the state-of-the-art. Our results show that, out of the four autoencoders, the denoising AE gave the best result in terms of average statistical similarity while the entity embedding AE was the best in terms of average machine learning efficacy. The algorithm implementations as well as the evaluation data are available on GitHub. [1]

1 Introduction

Generative modeling in machine learning (ML) focuses on learning domain representations with the objective of synthetically generating data that closely resembles the reality. The main idea behind generative modeling is to assume that real world observations come from a distribution $x \sim p(x)$. At the same time, the generative model can be viewed as an estimating distribution $q_\theta(x)$ described by a set of parameters θ. Then, we can formulate the objective of generative modeling as trying to ensure that $q_\theta(x)$ resembles $p(x)$ as closely as possible (using some statistical divergence: $D(p||q) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$). Data synthesis has been one of the most actively researched domains in ML in the recent past resulting in the creation of various state-of-the-art techniques such as GANs (Generative-Adversarial Networks) [5], VAE (Variational Autoencoders) [8] and LDMs (Latent Diffusion Models) [18]. Each of these techniques have been used to further create domain-efficient models for image-synthesis, text-synthesis as well as tabular data synthesis.

Tabular data synthesis involves generating a sequence of fake, columnar data that is statistically similar to the original table. The process typically requires representing categorical data as one-hot-encoded vectors and continuous data as normalized samples from multi-modal Gaussian distributions. Such representations add sparsity to the input data which have been shown to increase the sensitivity to column permutations [26] effectively reducing the quality of the synthetic data. Autoencoders provide an efficient way for data compression [6] and therefore are an ideal choice for reducing sparsity in tabular data.

[1] https://github.com/edithal-14/csc2516_2023_project

In this project, we focus on integrating variants of autoencoders in the generation of fake tabular data with the objective of assessing their individual impact on the quality of the synthetic data. We train the model on 5 different datasets, and compare the ML efficacy as well as statistical similarity of the synthetically generated data against the real data. The performances of proposed architectures are compared against current state-of-the-art models for tabular data synthesis.

2 Related Work

There have been many parametric copula-based approaches to tabular data synthesis in the past [20][19][13] that model the dependence structure between features with assumed independence and marginal distributions. However, these techniques are restricted by the underlying parametric assumptions of the distributions. Due to recent advancements in deep generative models and their various applications [11][1][3], there has been a growing interest in exploring GAN-based tabular data synthesis. Xu and Veeramachaneni [23] first introduced the framework of representing continuous variables in one-hot encoded vectors through *mode-specific normalization*, a process in which Gaussian mixture models are used to fit and normalize individual features. CTGAN (Xu et al. [24]) is a continuation of this work using variational Gaussian mixtures for mode-specific normalization, in addition to *training-by-sampling* for categorical variables, a process which ensures the training data can evenly explore all possible discrete values, and the addition of the *conditional vector* in order to encourage the model to synthesize desired discrete values. Park et al. [12] used an auxiliary classifier to ensure that the synthetically generated data is optimal for downstream ML tasks. CTAB-GAN (Zhao et al. [25]) combines the auxiliary classifier and the CTGAN framework, and adds encoding methods for mixed-type variables and transformations for encoding continuous data. However, all of these methods suffer from sparse representation of the input data, due to the mode-specific normalization and the conditional vector relying on one-hot encoding. AE-GAN (Zhu et al. [26]) deals with this issue by using an autoencoder to learn a more dense latent representation of the input data. The aim of this paper is to expand upon the idea of learning optimal latent representations of the input data. The objective is to generate data that demonstrates optimal ML efficacy while retaining statistical similarities to the original data. To achieve this, we have experimented with vanilla autoencoders [2], denoising autoencoders [22], and variational autoencoders [9]. Additionally, previous research has shown that entity embeddings can be used instead of one hot encoding for categorical variables to achieve better performance [7]. Thus, we also experimented with autoencoders that incorporate entity embeddings, which we refer to as entity embedding autoencoders.

3 Methodology

3.1 Architecture

The architecture used in this project involves 5 main components: the encoder $\mathcal{E}nc$, the decoder $\mathcal{D}ec$, the classifier $\mathcal{C}lf$, the generator $\mathcal{G}$, and the discriminator $\mathcal{D}$ which can be seen in Figure 1.

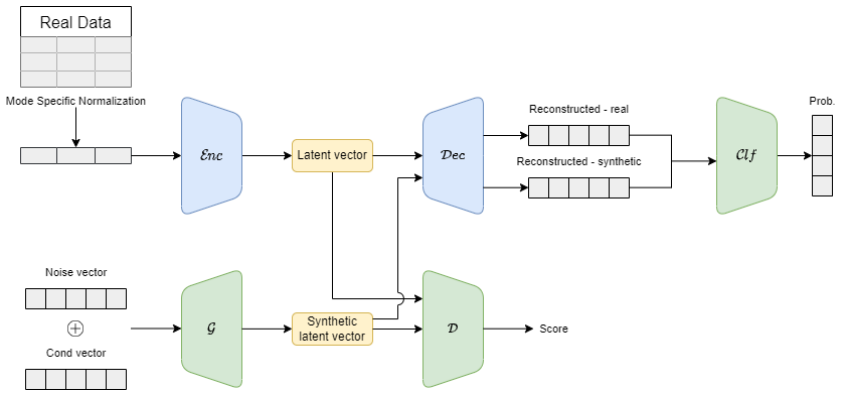

Figure 1: Overall architecture and data flow

The encoder and decoder are initially trained on the input data to learn a more compact latent representation. Once trained, the latent vectors can be used as real samples for the GAN, where the generator can directly generate latent vectors, and the discriminator can generate scores on real and synthetic latent vectors. Both the real and synthetic vectors can be decoded into the original data space using the decoder, to be fed into the classifier to ensure semantic integrity of the synthetic data.

3.2 Model Training

Prior to the GAN training, the autoencoder is train based on mean squared error (MSE) reconstruction loss, the loss function can be written as:

$$\mathbb{L}_{AE} = \mathbb{E}||\mathbf{x} - \hat{\mathbf{x}}||_2^2$$

where $\mathbf{x}$ and $\hat{\mathbf{x}}$ correspond to the input and the reconstructed input. Once the autoencoder is trained, the GAN can be trained based on generator loss, discriminator loss, and the classifier loss. Following the CTGAN framework [24], the Wasserstein loss with gradient penalty is used for the discriminator to differentiate between real and synthetic latent vectors. The generator loss is based on the discriminator feedback and the cross entropy loss from the classifier feedback on the synthetic latent vectors. The classifier is trained on both real and synthetically generated samples (as opposed to latent vectors) using cross entropy loss. The loss functions can be described as follows:

$$\begin{aligned}
\mathbb{L} &= \mathbb{L}_D + \mathbb{L}_G + \mathbb{L}_C \\
\mathbb{L}_D &= -\mathbb{E}\left[D(\mathbf{x}) - D(G(\mathbf{z}, cond)) - \lambda\left(||\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})||_2 - 1\right)^2\right] \\
\mathbb{L}_G &= -\mathbb{E}\left[D(G(\mathbf{z}, cond))\right] + CE(\hat{\mathbf{m}}, \mathbf{m}) \\
\mathbb{L}_C &= CE(\hat{y}^{(\mathbf{x})}, y) + CE(\hat{y}^{(\hat{\mathbf{x}})}, y)
\end{aligned}$$

where $\mathbf{z}$ is the noise vector, $\mathbf{m}$ is the mask vector corresponding to desired discrete values, $cond$ is the conditional vector corresponding to the mask vector [24], $\hat{\mathbf{m}}$ is the vector corresponding to the discrete values generated by $\mathcal{G}$, $\hat{y}^{(\mathbf{x})}$ is the prediction generated by $\mathcal{C}lf$ based on $\mathbf{x}$, and $\hat{y}^{(\hat{\mathbf{x}})}$ is the prediction generated by $\mathcal{C}lf$ based on $\hat{\mathbf{x}}$. The training process for the architecture is detailed in Algorithm 1.

Algorithm 1 Training Algorithm

Input: training data $\mathbf{T}_{train}$, parameters $\boldsymbol{\theta}_G$, $\boldsymbol{\theta}_D$, $\boldsymbol{\theta}_C$, pre-trained encoder and decoder, pac value pac, loss criteria $\mathbb{L}_D$, $\mathbb{L}_G$, $\mathbb{L}_C$

Create conditional vector $cond$ based on desired discrete mask vector $\mathbf{m}$
Sample $\mathbf{z} \sim \texttt{MVN}(0, \mathbf{I})$
$\hat{\mathbf{v}} \leftarrow \texttt{Generator}(\mathbf{z}, cond)$
Sample real data $\mathbf{x} \sim \texttt{Uniform}(\mathbf{T}_{train}|cond)$
$\mathbf{v} \leftarrow \texttt{Encoder}(\mathbf{x})$
$\mathcal{L}_D \leftarrow \mathbb{L}_D(pac, \mathbf{m}, \mathbf{v}, \hat{\mathbf{v}})$
Backpropagate $\mathcal{L}_D$ and update discriminator weights
Regenerate $\hat{\mathbf{v}} \leftarrow \texttt{Generator}(\mathbf{z}, cond)$
$\hat{\mathbf{x}} \leftarrow \texttt{Decoder}(\hat{\mathbf{v}})$
Obtain mask vector $\hat{\mathbf{m}}$ corresponding to discrete values of $\hat{\mathbf{x}}$
$\mathcal{L}_G \leftarrow \mathbb{L}_G(\hat{\mathbf{v}}, \mathbf{m}, \hat{\mathbf{m}})$
Backpropagate $\mathcal{L}_G$ and update generator weights
Obtain real label y and prediction $\hat{y}^{(\mathbf{x})}$ corresponding to $\mathbf{x}$
Obtain prediction $\hat{y}^{(\hat{\mathbf{x}})}$ corresponding to $\hat{\mathbf{x}}$
$\mathcal{L}_C \leftarrow \mathbb{L}_C(y, \hat{y}^{(\mathbf{x})}, \hat{y}^{(\hat{\mathbf{x}})})$
Back propagate $\mathcal{L}_C$ and update classifier and generator weights

3.3 Evaluation Methods

The evaluation of the GAN models is based on two crietria: ML efficacy, and statistical similarity scores. The ML efficacy is measured by comparing the accuracy, F1-score, and the AUC obtained

from a model trained on real data against the same metrics obtained from synthetic data, where both models are evaluated against the same testing data. To obtain fair model-agnostic metrics, 5 distinct individual classifiers (logistic regression, support vector machine, decision tree, random forest and multi-layer perceptron with 100 neurons and one layer) will be used to compute each of the metrics, and the resulting mean values will be used. Statistical similarity scores include the Jensen-Shannon divergence (JSD) [10], Wasserstein distance (WD) [15], and the difference in pair-wise correlation (diff. corr.) [25]. The experiments will be carried out on 5 datasets: Adult[4], Loan[14], Credit[21], Intrusion[16] and Covertype[17]. Refer to Table 1 for a description of the datasets.

Table 1: Description of datasets

Name	#Train/Test	Target	#C	#B	#M
Adult	39k/9k	"income"	5	2	7
Covertype	45/5k	"Cover_type"	10	44	1
Credit	40k/10k	"Class"	30	1	0
Intrusion	45k/5k	"Class"	24	6	14
Loan	4k/1k	"Personal Loadn"	6	5	2

#C, #B, #M correspond to number of continuous, binary, and multi-class discrete columns respectively.

4 Results

Refer to Table 7 for the ML efficacy and statistical similarity scores of each of the architectures averaged over 3 replications for all datasets.

Table 2: ML efficacy difference and statistical similarity scores averaged over 5 datasets with 3 replications

Method	Accuracy	F1-score	AUC	JSD	WD	Diff. Corr.
CTAB-GAN	**9.83%**	**0.127**	**0.117**	**0.069**	1.050	**2.10**
CTGAN	21.51%	0.274	0.253	0.070	1.769	2.73
Vanilla AE-GAN	29.37%	0.436	0.450	0.226	**0.085**	6.28
Denoising AE-GAN	30.54%	0.490	0.295	0.330	0.150	5.89
Entity AE-GAN	18.89%	0.378	0.351	0.266	0.098	6.57
Variational AE-GAN	25.26%	0.438	0.387	0.279	0.097	10.94

For the Adult dataset, denoising AE-GAN had the best accuracy and AUC but worse F1 score compared to the vanilla AE-GAN. For the Credit dataset, denoising AE-GAN had the best statistical similarity and ML efficacy values. For the Loan dataset, all architectures performed comparably, with denoising AE-GAN and variational AE-GAN having worse accuracy then the other two AEs. For the Intrusion dataset, entity embedding AE-GAN had the best ML efficacy values but performed worse in statistical similarity where the vanilla AE-GAN performed the best. For the Covertype dataset, entity embedding AE-GAN had the best ML efficacy values but worse statistical similarity compared to vanilla AE-GAN. For details on the individual results refer to Appendix B. When averaged across all datasets, **entity embedding AE-GAN performed the best in 2 out of 3 ML efficacy metrics**. Upon performing error analysis, it was found that the vanilla AE-GAN produced semantically incoherent rows where sex was male and relationship was wife, which can likely cause decrease in ML efficacy as well as statistical similarities. Variational AE-GAN was prone to mode collapse, producing the same rows for in each dataset across most datasets, excluding the Covertype dataset. This is evident from the high correlation coefficients across all datasets for Variational AEGAN.

5 Conclusion

This project aimed to leverage autoencoders to find denser latent representations of tabular data in order to assist the learning process of GANs. However, our results show that the benchmark methods (CTGAN [24] and CTAB-GAN [25]) which do not use AEs had better statistical similarity and ML efficacy metrics. We attribute this observation to mode collapse in the GAN training process assisted by AEs. Our models produced similar (in some cases exact) rows of data which significantly affected the learning ability of classifiers.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[2] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, 2021.

[3] Adriel Cheng. Pac-gan: Packet generation of network traffic using generative adversarial networks. In *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 0728–0734, 2019. doi: 10.1109/IEMCON.2019.8936224.

[4] Dheeru Dua and Casey Graff. UCI machine learning repository, 2019. URL `https://archive.ics.uci.edu/ml/datasets/Adult`.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[6] Karol Gregor, Frederic Besse, Danilo Jimenez Rezende, Ivo Danihelka, and Daan Wierstra. Towards conceptual compression. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL `https://proceedings.neurips.cc/paper_files/paper/2016/file/4abe17a1c80cbdd2aa241b70840879de-Paper.pdf`.

[7] Cheng Guo and Felix Berkhahn. Entity embeddings of categorical variables. *CoRR*, abs/1604.06737, 2016. URL `http://arxiv.org/abs/1604.06737`.

[8] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[9] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL `http://arxiv.org/abs/1312.6114`.

[10] J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991. doi: 10.1109/18.61115.

[11] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

[12] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. Data synthesis based on generative adversarial networks. *Proceedings of the VLDB Endowment*, 11(10):1071–1083, jun 2018. doi: 10.14778/3231751.3231757. URL `https://doi.org/10.14778%2F3231751.3231757`.

[13] Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. The synthetic data vault. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 399–410, 2016. doi: 10.1109/DSAA.2016.49.

[14] Pritech. Bank personal loan modelling dataset, 2021. URL `https://www.kaggle.com/code/pritech/bank-personal-loan-modelling/data`.

[15] Aaditya Ramdas, Nicolas Garcia, and Marco Cuturi. On wasserstein two sample testing and related families of nonparametric tests, 2015.

[16] UCI Machine Learning Repository. KDD Cup 1999 Data, 1999. URL `http://archive.ics.uci.edu/ml/datasets/kdd+cup+1999+data`.

[17] UCI Machine Learning Repository. Covertype data set, 2019. URL `https://archive.ics.uci.edu/ml/datasets/Covertype`.

[18] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

[19] Thorsten Schmidt. Coping with copulas. *Copulas - From Theory to Application in Finance*, 01 2007.

[20] Yi Sun, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Learning vine copula models for synthetic data generation. AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33015049. URL https://doi.org/10.1609/aaai.v33i01.33015049.

[21] MLG ULB. Credit card fraud detection dataset, 2016. URL https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud.

[22] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning proceedings*. 2008.

[23] Lei Xu and Kalyan Veeramachaneni. Synthesizing tabular data using generative adversarial networks. *CoRR*, abs/1811.11264, 2018. URL http://arxiv.org/abs/1811.11264.

[24] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional gan. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/254ed7d2de3b23ab10936522dd547b78-Paper.pdf.

[25] Zilong Zhao, Aditya Kunar, Hiek Van der Scheer, Robert Birke, and Lydia Y. Chen. CTAB-GAN: effective table data synthesizing. *CoRR*, abs/2102.08369, 2021. URL https://arxiv.org/abs/2102.08369.

[26] Yujin Zhu, Zilong Zhao, Robert Birke, and Lydia Y. Chen. Permutation-invariant tabular data synthesis, 2022. URL https://arxiv.org/abs/2211.09286.

Appendix A: Contributions

Jimmy Woo: Ideation of research topics, literature review, implementation of AE-GAN framework and all variants of autoencoders, implementation of model performance evaluation pipeline, and work on final report

Sumant Bagri: Ideation of research topics, literature review and designing the framework for hyper-parameter tuning for different datasets. Analysis of the results and compiling the proposal and final report.

Vignesh Edithal: Identifying datasets for evaluation and performing data cleaning and pre-processing. Setting up infrastructure to parallelize hyper-parameter tuning on SLURM cluster. Performing error analysis and writing the results and conclusion section of the report.

Appendix B: Individual Results

Note that the authors of CTAB-GAN paper [25] do not provide dataset-wise results for CTGAN and CTAB-GAN models. We are omitting these models in the tables below.

Method	Accuracy	F1-score	AUC	JSD	WD	Diff. Corr.
Vanilla AE-GAN	18.57%	0.284	0.32	0.389	0.074	2.30
Denoising AE-GAN	9.34%	0.329	0.274	0.547	0.115	2.62
Entity AE-GAN	9.97%	0.316	0.287	0.455	0.111	4.62
Variational AE-GAN	10.62%	0.335	0.574	0.495	0.074	7.47

Table 3: Results on adult dataset

Method	Accuracy	F1-score	AUC	JSD	WD	Diff. Corr.
Vanilla AE-GAN	24.97%	0.474	0.606	0.028	0.031	6.83
Denoising AE-GAN	0.08%	0.350	0.304	0.028	0.029	2.76
Entity AE-GAN	8.80%	0.391	0.591	0.086	0.040	6.75
Variational AE-GAN	24.99%	0.474	0.445	0.028	0.043	22.51

Table 4: Results on credit dataset

Method	Accuracy	F1-score	AUC	JSD	WD	Diff. Corr.
Vanilla AE-GAN	9.55%	0.359	0.264	0.405	0.163	3.03
Denoising AE-GAN	10.75%	0.335	0.248	0.408	0.163	2.99
Entity AE-GAN	8.99%	0.354	0.251	0.409	0.162	3.04
Variational AE-GAN	10.52%	0.365	0.273	0.409	0.162	3.05

Table 5: Results on loan dataset

Method	Accuracy	F1-score	AUC	JSD	WD	Diff. Corr.
Vanilla AE-GAN	56.24%	0.648	0.652	0.204	0.072	13.00
Denoising AE-GAN	55.70%	0.639	0.246	0.489	0.243	13.51
Entity AE-GAN	30.31%	0.405	0.256	0.226	0.070	12.16
Variational AE-GAN	42.96%	0.586	0.254	0.313	0.095	14.92

Table 6: Results on intrusion dataset

Method	Accuracy	F1-score	AUC	JSD	WD	Diff. Corr.
Vanilla AE-GAN	37.51%	0.414	0.407	0.107	0.084	6.23
Denoising AE-GAN	76.83%	0.795	0.407	0.179	0.202	7.56
Entity AE-GAN	36.42%	0.427	0.370	0.151	0.108	6.72
Variational AE-GAN	37.21%	0.432	0.388	0.151	0.108	6.75

Table 7: Results on covertype dataset