Diffusion Models for Data Imputation

Final Project Report

Name: Johannes Fuest
SUNet ID: jfuest
Department of Statistics
Stanford University
`jfuest@stanford.edu`

Name: Linus A. Hein
SUNet ID: lhein
Department of Electrical Engineering
Stanford University
`lhein@stanford.edu`

Name: Alex Thiesmeyer
SUNet ID: thiesmey
Department of Statistics
Stanford University
`thiesmey@stanford.edu`

1 Introduction

Many modern machine learning methods require large amounts of data. In many applications, the acquisition of such large amounts of data presents a challenge. Exacerbating this challenge, often the data we are able to collect does not conform to the exact specifications of our model. That is, certain features which our model expects as input may be missing.

In order to avoid having to ignore these valuable but incomplete samples, we can estimate missing values using a data imputation method. Imputation is commonly done using methods which can be implemented quickly and easily, like mean/mode imputation and random forest imputation. More complex methods, namely generative methods like variational autoencoders (VAEs) and generative adverserial models (GANs)[1], have also been shown to achieve success when applied to data imputation problems.

Diffusion models are a trending generative technique that operate differently than VAEs and GANs[2]. Diffusion models essentially learn how to reconstruct a data point of a target distribution from noise. While this type of model has been most successful in image domains, Kotelnikov et al., 2022 introduced the TabDDPM model to apply the diffusion process to the generation of synthetic tabular data[3]. In this project, we adapted TabDDPM for data imputation.

2 Related Work

Though diffusion models were first introduced in 2015 and have been used in a variety of different applications, from image synthesis and video generation to molecular design, their application to tabular data imputation is new and no directly related work exists [2] [4]. We found only one, unpublished, instance of an attempt to use diffusion models for data imputation, with no available data or code [5]. There have been positive results from efforts to include diffusion models in time series data imputation in combination with other models[6], but this has not been attempted for tabular data.

Other related data imputations methods have been the subject of academic debate for multiple decades[7]. Broadly speaking, two major categories of data imputation methods: statistical methods, such as mean/mode imputation, and machine learning based techniques, such as k-NN imputation [8] [9]. There is no universally superior method, with the quality of imputations dependent on characteristics of the underlying data, as well as its missingness status, i.e. whether the data is missing completely at random (MCAR), missing not at random (MNAR) or missing at random (MAR) [10]. In practice, a wide range of imputation methods are used, with even the basic mean/mode methods seeing plenty of use due to their simplicity [7]. One class of methods closely related to our approach is the adaptation of GANs to imputation[11]. Different variations of GANs have been used successfully for data imputation over the past years [12]. Given that diffusion models have recently outperformed GANs in image sythesis, one of its most common use cases [13], we now investigate whether they can replicate this performance in the new domain of tabular data imputation.

3 Methods

3.1 The TabDDPM Model

The high level structure of the TabDDPM model is shown in Figure 1. The model takes one-hot encoded categorical features and normalized numerical features as input [3]. The model is evaluated by an ML efficiency metric, which compares how well a learning algorithm performs after training on synthetic data generated by the model to how well the algorithm performs using true training data [3]. As such, the hyperparameters of TabDDPM are tuned based on the performance of a downstream task (classification or regression) particular to each dataset [3]. The authors argue that using the CatBoost learning algorithm for tuning produces the most accurate synthetic data and that doing so does not bias the model to producing data which favors the CatBoost model specifically [3].

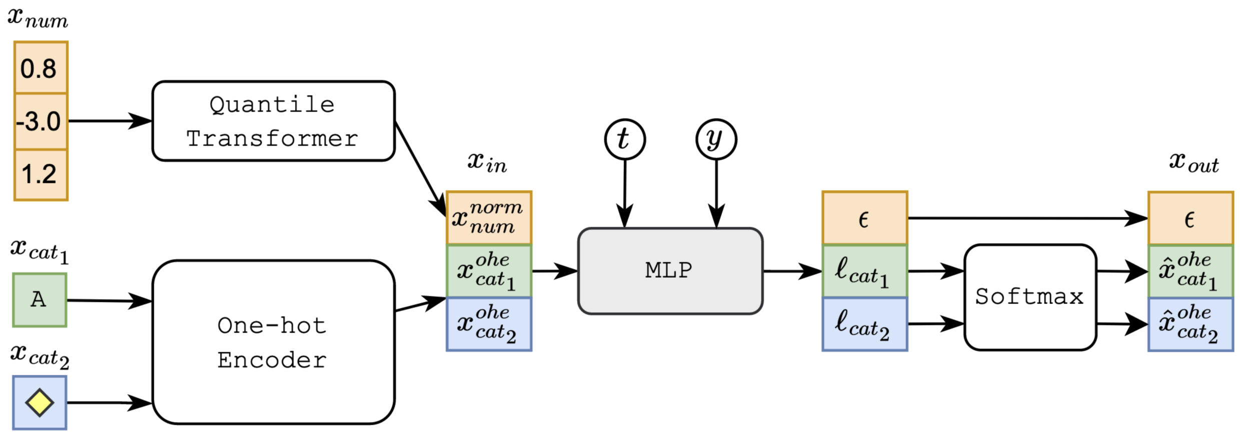


Figure 1: TabDDPM scheme for classification problems; t, y, and l denote a diffusion timestep, a class label, and logits, respectively. From [3].

3.2 Our approach

Model Training

We accepted the authors' argument regarding CatBoost, and, for our analysis, used the CatBoost algorithm to tune the hyperparameters of the TabDDPM model for each dataset. The TabDDPM model was then trained in the same manner for our method as for synthetic data generation.

Data Normalization

The authors of TabDDPM used quantile normalization to transform every numerical feature to be as close as possible to a normal distribution. This is important, as the formulation of TabDDPM assumes that all numerical features follow a gaussian distribution. However, the specific implementation from the scikit-learn library used in the paper has trouble mapping numerical distributions with few observed unique values to gaussians. For this reason, before fitting the quantile transformation on the training data, we add a small amount of noise to the training data to spread out replicate values, allowing the quantile transformer to perform better, as seen in Fig. 2, leading to a small improvement in model performance.

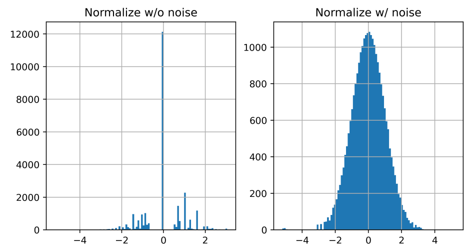


Figure 2: Adding a small amount of noise improved normalization performance

Data Imputation

To simulate missing data, we applied a mask to a column of our testing set and filled masked values with NaN values. The goal was to fill in the missing NaN values. To understand how we used the TabDDPM model to achieve this goal, it is instructional to take a brief look at how the authors of TabDDPM model their diffusion process. For numerical features x^n, they write [3]:

$$\begin{aligned} q(x_t^n|x_{t-1}^n) &:= \mathcal{N}\left(x_t^n; \sqrt{1-\beta_t}x_{t-1}^n, \beta_t I\right) \\ q(x_T^n) &:= \mathcal{N}(x_T^n; 0, I) \\ p_\theta(x_{t-1}^n|x_t^n) &:= \mathcal{N}(x_{t-1}^n; \mu_\theta(x_t^n, t), \Sigma_\theta(x_t^n, t)) \end{aligned}$$

For categorical features x^c [3]:

$$\begin{aligned} q(x_t^c|x_{t-1}^c) &:= Cat\left(x_t^c; (1-\beta_t)x_{t-1}^c + \beta_t/K\right) \\ q(x_T^c) &:= Cat(x_T^c; 1/K) \end{aligned}$$

For synthetic data generation, we would begin by choosing some random vector x_T whose numerical features are sampled from $\mathcal{N}(x_T^n; 0, I)$, and whose categorical features are sampled from $Cat(x_{T,cat}^c; 1/K)$. We would then feed $x_{t=T}$ and the timestep t into the neural network, which produces the parameters of the probability distributions from which to sample x_{t-1}. In other words: $x_{t-1} \sim p_\theta(x_{t-1}|x_t, t)$. Repeating this process until $t = 0$ would then yield a generated sample.

For imputation, our input is a given row of data with some values known and some unknown, not a random vector. From this incomplete vector input, we want to generate a sample that matches the known values and produces reasonable estimations of missing values. We found the most effective approach for this to be the following: we use the known forward diffusion model $q(x_t|x_0, t)$ to replace our known values at time t, and the learned diffusion model $p_\theta(x_{t-1}|x_t, t)$ to fill in the unknown variables at time T that we wish to impute. Programmatically, we can write:

$$x_{t,i} = \begin{cases} q(x_t|x_0, t)_i & \text{if } x_{0,i} \neq \texttt{NaN} \\ p_\theta(x_t|x_{t+1}, t+1)_i & \text{if } x_{0,i} = \texttt{NaN} \end{cases} \tag{1}$$

Since only $p_\theta(x_{t-1}|x_t, t)$ is given by our learned diffusion model, we still have to define $q(x_t|x_0, t)$. For numerical features x_t^n, we used a function `gaussian_q_sample` implemented by the TabDDPM authors which samples

$$x_t^n \sim \mathcal{N}\left(x_0^n\sqrt{\prod_{i=1}^{t}(1-\beta_i)}, 1 - \prod_{i=1}^{t}(1-\beta_i)\right).$$

In our experiments we found no noticeable difference in imputation quality when changing variance to zero, i.e., when sampling

$$x_t^n \sim \mathcal{N}\left(x_0^n\sqrt{\prod_{i=1}^{t} 1-\beta_i x_0^n}, 0\right)$$

or in other words setting $x_t^n := x_0^n\sqrt{\prod_{i=1}^{t} 1-\beta_i}$ [1]. Given that both approaches yielded similar results, we decided to use zero variance, because adding noise to the known variables is equivalent to taking information away from the diffusion model.

For the categorical features we again tried using a pre-implemented function `q_sample`, which sets

$$x_t^c \sim Cat\left(x_t^c; \left(\prod_{i=1}^{t}(1-\beta_i)\right)x_0^c + \left(1 - \left(\prod_{i=1}^{t}(1-\beta_i)\right)\right)/K\right)$$

However, we saw similar performance by using a simpler method, setting $x_t^c := x_0^c$.

[1]Setting $x_t^n := x_0^n$ lead to significantly worse performance, which may have to do with the neural network being trained to expect smaller values of x_t^n for larger values of t.

4 Experiments

Datasets

Our method was applied to a total of sixteen datasets. Fifteen of these were used in Kotelnikov et al., 2022 to test the performance of synthetic data generation using TabDDPM [3]. The hyperparameters of the TabDDPM model had already been tuned on these datasets, allowing for an expedited training process. We imported and tuned the hyperparameters of one additional dataset, JapaneseVowels, to ensure that we could apply our method to datasets as yet unseen by the TabDDPM model.

Experimental Process

Once the model was trained on a dataset, a random feature was chosen to be imputed. Twelve different experiments were run.

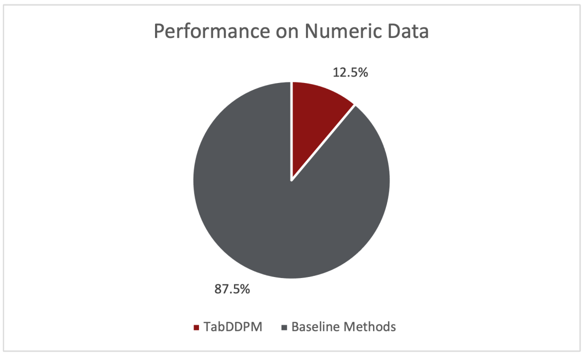
First, a mask was generated for the test data of the chosen feature using either a MCAR, MNAR, or MAR missingness pattern. Following the example of Jäger et al., 2021, the implementation of these missingness patterns was adapted from the jenga Python package developed by Sebastian Schelter [1][14]. For MNAR patterns, a random lower percentile was generated and a higher percentile was chosen according to the proportion of missing data desired. Values of the feature lying between these percentiles were masked. For MAR patterns, the values of another randomly selected column which were between the percentiles determined the values to be masked in the chosen feature. For each missingness pattern, the method was tested with a proportion of missing data of 0.01, 0.1, 0.3, and 0.5.

In addition to our method, each experiment was run using two baseline data imputation approaches. For numerical features, experiments were run substituting masked values with the mean of the feature in the training set, and by using a random forest regressor trained on the training set [15]. Similarly, categorical features had masked values replaced with the mode class in the training data, and with the output of a random forest classifier trained on the training set [15]. No preprocessing was used for these methods.

The root mean squared error between imputed values and true values was used to evaluate a method's performance on numerical features. On categorical features, the macro F1 score was used.

5 Results

As shown in Fig. 3, our method was consistently outperformed by baseline imputation methods on numerical features. Our method performed best in only 15 out of 120 numeric data imputation experiments we conducted. The method performed significantly better on categorical data, outperforming random forest and mean/mode imputation in 42 out of 72 experiments conducted with categorical data. In Table 1 we only show the results for 10% missing data, but they are similar to those we observed for other levels of missing data. We found no evidence that our method performed differently based on the proportion of missing data.



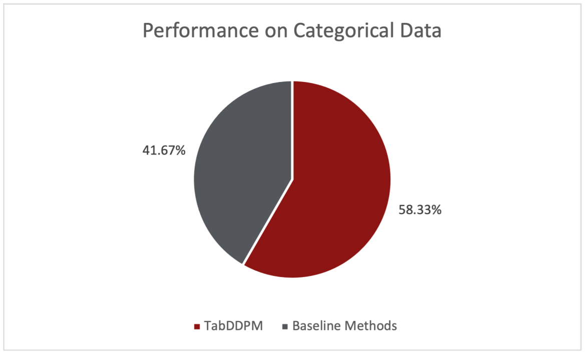


Figure 3: TabDPPM outperformance on numeric vs on categorical data

Dataset	Abalone			Adult			Buddy		
	MCAR	MAR	MNAR	MCAR	MAR	MNAR	MCAR	MAR	MNAR
Mean/Mode	0.1818	0.2727	**1.0000**	12.0485	11.3982	14.3676	0.1923	0.2007	0.1837
Random Forest	**0.7190**	0.3429	0.3846	**10.3463**	**10.5234**	**12.1608**	0.5487	0.4719	0.5606
TabDDPM	0.1905	**0.3636**	0.1937	14.2834	14.9314	15.5247	**0.7513**	**0.6898**	**0.7473**

Dataset	California			Cardio			Churn2		
	MCAR	MAR	MNAR	MCAR	MAR	MNAR	MCAR	MAR	MNAR
Mean/Mode	0.4042	0.1129	0.0755	0.3890	0.3873	0.3880	0.3377	0.3443	1.0000
Random Forest	**0.1108**	**0.0718**	**0.0458**	**0.7373**	**0.7257**	**0.7693**	0.4479	0.5886	0.4048
TabDDPM	0.3790	0.1144	0.0652	0.6214	0.6040	0.6997	**1.0000**	**1.0000**	**1.0000**

Dataset	Diabetes			FB Comments			Gesture		
	MCAR	MAR	MNAR	MCAR	MAR	MNAR	MCAR	MAR	MNAR
Mean/Mode	10.3491	**6.8101**	**3.4343**	117.8655	109.3840	30.3612	0.0008	0.0018	0.0013
Random Forest	9.2168	7.2049	4.9026	**2.8292**	**3.0761**	**0.9898**	**0.0006**	**0.0010**	0.0010
TabDDPM	**8.8205**	10.0074	11.2128	35.4298	32.3356	5.4699	0.0006	0.0014	**0.0010**

Dataset	Higgs Small			House			Insurance		
	MCAR	MAR	MNAR	MCAR	MAR	MNAR	MCAR	MAR	MNAR
Mean/Mode	0.5225	0.4699	0.4480	0.3200	0.2890	0.3877	0.1061	0.0667	0.0000
Random Forest	**0.3858**	**0.3431**	**0.3100**	**0.3014**	**0.2620**	**0.3265**	0.4083	0.3330	0.2690
TabDDPM	0.5496	0.5039	0.4229	0.3767	0.3288	0.4431	**0.5954**	**0.5868**	**0.3697**

Dataset	King			Miniboone			Wilt		
	MCAR	MAR	MNAR	MCAR	MAR	MNAR	MCAR	MAR	MNAR
Mean/Mode	0.4983	0.4983	1.0000	55.3293	3.4683	3.4492	33.5430	84.9879	23.3350
Random Forest	**0.7488**	**0.6983**	**1.0000**	**0.0555**	**0.0538**	**0.0355**	**8.1695**	**15.4178**	**5.3480**
TabDDPM	0.4977	0.6620	0.4988	36.7834	0.0866	0.0568	12.3297	48.2451	7.4212

Dataset	Japanese Vowels		
	MCAR	MAR	MNAR
Mean/Mode	0.3172	0.2678	0.1300
Random Forest	**0.0916**	**0.0797**	**0.0812**
TabDDPM	0.1016	0.0959	0.0943

Table 1: Performance with 10% missing data. Abalone, Buddy, Cardio, Churn2, Insurance, and King show macro F1 scores (higher is better). The rest show RMSE (lower is better).

6 Discussion

Given that our method was able to outperform the mean/mode method of imputation in many experiments, we have at least demonstrated that it is feasible for data imputation. However, one clear conclusion from our results is that our method should not be used on numeric features. Tuning the hyperparameters of the TabDDPM model on a new dataset took ten to twenty hours. The random forest baseline, on the other hand, could impute values on a newly provided dataset in less than a minute and with a higher degree of accuracy in most cases. Our results align with the results of the TabDDPM model for synthetic data generation, which performed comparably to the simpler SMOTE technique [3]. Unfortunately, the advantages for privacy which TabDDPM exhibited for synthetic data do not translate to imputation problems.

The evidence is more promising for categorical data, where our method was able to outperform the simpler baseline methods more consistently. However, when time considerations are weighed, even these positive results fail to justify regular use of our method.

7 Future Work

With that said, if the tuning time of the TabDDPM model were to be shortened drastically, continued investigation might be worthwhile. In particular, further testing on categorical features could be conducted to determine if the method actually does represent an improvement in that area. Results would need to be compared to a wider selection of generative and discriminative imputation techniques to draw such a conclusion. Also, it might be worth studying the performance of our method after the TabDDPM model is trained on incomplete training data. We originally wanted to test this capability, but had to prioritize other tasks due to the chokehold that tuning time placed on our work.

8 Contributions

Johannes Fuest: Results and error analysis, related work, report, references, poster
Alex Thiesmeyer: Wrote the code underlying the experimental process and the importation of new datasets. Ran experiments.
Linus A. Hein: Wrote the code for data imputation, and for our change to data normalization. Wrote section 3.2. Ran JapaneseVowels experiment.

Our code can be found here

References

[1] Sebastian Jäger, Arndt Allhorn, and Felix Bießmann. A benchmark for data imputation methods. *Frontiers in big Data*, page 48, 2021.

[2] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *CoRR*, abs/1503.03585, 2015.

[3] Akim Kotelnikov, Dmitry Baranchuk, Ivan Rubachev, and Artem Babenko. Tabddpm: Modelling tabular data with diffusion models. *arXiv preprint arXiv:2209.15421*, 2022.

[4] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Yingxia Shao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications. *arXiv preprint arXiv:2209.00796*, 2022.

[5] Shuhan Zheng and Nontawat Charoenphakdee. Diffusion models for missing value imputation in tabular data. *arXiv preprint arXiv:2210.17128*, 2022.

[6] Juan Miguel Lopez Alcaraz and Nils Strodthoff. Diffusion-based time series imputation and forecasting with structured state space models. *arXiv preprint arXiv:2208.09399*, 2022.

[7] Wei-Chao Lin and Chih-Fong Tsai. Missing value imputation: a review and analysis of the literature (2006–2017). *Artificial Intelligence Review*, 53(2):1487–1509, 2020.

[8] Tero Aittokallio. Dealing with missing values in large-scale studies: microarray data imputation and beyond. *Briefings in bioinformatics*, 11(2):253–264, 2010.

[9] Pedro J García-Laencina, José-Luis Sancho-Gómez, and Aníbal R Figueiras-Vidal. Pattern classification with missing data: a review. *Neural Computing and Applications*, 19(2):263–282, 2010.

[10] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019.

[11] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

[12] Jaeyoon Kim, Donghyun Tae, and Junhee Seok. A survey of missing data imputation using generative adversarial networks. In *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 454–456. IEEE, 2020.

[13] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.

[14] Sebastian Schelter. Jenga. `https://github.com/schelterlabs/jenga`, November 2021. Accessed: 2022-12-08.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[16] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[17] Ramiro Daniel Camino, Christian Hammerschmidt, et al. Working with deep generative models and tabular data imputation. 2020.

[18] Ramiro D. Camino, Christian A. Hammerschmidt, and Radu State. Improving missing data imputation with deep generative models. 2019.

[19] Github User "gruns". Python icecream library. `https://github.com/gruns/icecream`, 2022. Accessed: 2022-12-08.

[20] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[21] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[22] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[23] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[24] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[25] Github User "hukkin". tomli python library. `https://github.com/hukkin/tomli`, 2022. Accessed: 2022-12-08.

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.