Missing data imputation using Autoencoders and Variational Autoencoders

By Boyang Lyu and Fangfang Qu

0. Abstract

Missing data is a common appearance that can be seen in almost all statistical analyses. Sometimes missing data can introduce bias and affect the results significantly. Though many data imputation methods are proposed previously, it's hard to determine which one is better to use in practice. In this paper, we compare four imputation methods including mean-replacement, K-nearest neighbour, Autoencoder, and Variational Autoencoder. We test these imputation methods on three datasets and found that for dataset with small-size features, both KNN and Variational Autoencoder(VAE) perform better than the other two. Furthermore, we find that VAE achieves the best performance for all kinds of cases.

1. Motivation

Missing data is a common appearance that can be seen in almost all statistical analyses. It can arise from a participant dropout during test, mistakes in data collections or the equipment defect, or limited time and financial cost for collecting all possible data [1-2]. Missing data can sometimes affect the conclusions significantly and produce biased estimates. Though discard incomplete data is a simple way to solve this problem, it leads to information loss and may shrink the sample size dramatically. Imputation is an alternate in handling missing data. By replacing the missing values with an estimation, we can manage to retain more information that contained in original dataset. Most commonly used methods such as mean-replacement and K-Nearest Neighbors (KNN) can achieve good performance. But the former often achieved accompanied with variance decrease, the latter will bear an increasing cost to finish a series of new imputations [3].

In this project, we are going to explore different methods and their performances on imputing missing data, for example, imputing with mean, encoding and decoding the whole dataset and get an estimate of missing data, using its K-nearest neighbors to locate the missing value and so on. Depending on these strategies, we implement our algorithm by following models: Variational Autoencoders (VAE), Autoencoder(AE), K-nearest neighbors(KNN) and mean replacement [3-6]. For the first three methods, we will train our models based on the complete data only; at the imputation stage, we first fill the missing position with mean and then refine the imputation by running our trained model to encode the mean replaced data, and decode to generate predictions. For KNN, if data is lightly corrupted, which means that only a few features for each instance are missing, we will compute weighted Euclidean distances (details in Method section) to determine missing values. All complete and incomplete data are used in KNN imputation.

Though all these four methods have been proposed for a long time, we can hardly find a comparison among them, also it's sometimes hard to determine which one is better to use in practice. Thus, we decide to explore this field and try to find an answer. Moreover, we will use complete datasets and randomly pick out data points to generate the corrupted dataset. Since ground truth is known, we can easily evaluate the performances of different methods by measuring mean square error between the reconstructed data and the true removed data or the classification accuracy.

Before doing experiments, we hypothesize that VAE will give us the best performance for both large and small size dataset, while KNN method may also work well on dataset with small size features. The reason for our first hypothesize is that the VAE fully specifies a generative distribution over the objects it is representing. Sometimes the remaining complete data may not share the same distribution as the missing data, AE thus is not able to make a good prediction after fully trained. However, things are different for VAE. Instead of predicting exact values, VAE generate a distribution for each missing value and thus has chances to cover the truth. We made the second

hypothesize based on fact that a point value can be approximated by the values of the points that are closest to it, which is what KNN does. But KNN may only work for datasets with small feature size because for high dimension data, distance is more likely to be equally large or small.

2. Methods

As mentioned above, we will mainly focus on the VAE model which we think will achieve our goal best and implement three other models to do the comparison. The model structure we will use for VAE is Bayesian Neural Network and the algorithm will be variational inference. For AE, we will simply use the autoencoder to do the point estimation without uncertainty. Algorithm and implementation will be described in detail below. We will choose mean replacement as our baseline.

1) Mean replacement

Algorithm:

1. Ignore all missing data
2. Calculate the mean of the remaining data with respect to each feature and replace the missing data with the calculated mean

2) Mean replacement plus VAE

The prior and likelihood for our VAE is as follow [4]:

Prior on Codes z:

$$p(z_n) = Normal(z_n|0, I) \tag{1}$$

Likelihood of Data given Code (for continuous data):

$$p(x_n|z_n,\theta) = Normal(x_n|ReLu(Lin(z_n,\theta)), \sigma^2 I) \tag{2}$$

where ReLu stands for rectified linear unit function and Lin stands for linear function

Approximate Posterior:

$$logq_{\phi}(z|x^{(i)}) = logN(z; \mu^{(i)}, \sigma^{2(i)} I) \tag{3}$$

Loss function [4]:

$$L(\theta,\varphi;x) = -D_{KL}(q_{\varphi}(z|x) \,||\, p_{\theta}(z)) + E_{q\varphi(z|x)}\, logp(x|z) \tag{4}$$

Reconstructed probability vector $\tilde{x}$

Given an input data vector x_n, we can "reconstruct" it by [4]:

$$\tilde{x}_n = \text{decode}(\text{encode}(x_n, \phi), \theta) \tag{5}$$

For optimizer algorithm, we will use ADAM [7] optimizer to achieve good results fast.

Algorithm:

We will first train a VAE model using fully observed data, then input the corrupted data with each missing position replaced with the mean of its column [8]. To mimic the situation the model will meet during imputation, we will also manually corrupt the training data, which will be described below.

Train:

1. Remove all missing row to get fully observed data
2. Manually corrupt the remaining data as training data

a. Calculate the missing rate from the removed incomplete data
b. Choose the same portion of data from the remaining dataset and replace with 'nan'
c. Replace 'nan' with mean calculated with respect to each feature(without count 'nan')

3. Train a VAE using the dataset described above

Imputation:

1. Consider removed incomplete data as test data
2. Fill the blank position with mean
3. Input the data to the trained VAE
4. Sample from the latent variable distribution (the output of the encoder) to generate z, given x
5. Sample from the reconstructed data distribution (the output of the decoder) to generate $\tilde{x}$ given z
6. Replace the missing values with the reconstructed values, leaving the observed values unchanged
7. Go back to step 3 for certain(here we use 25) times

Hyperparameters:

Learning rate, training epoch, number of impute iterations, neural network structures including hidden layers for both encoder and decoder, hidden units for each layer and batch size.

Strategy for tuning hyperparameters:

One thing need to mention is the choice of batch size. When dataset is heavily corrupted, a smaller batch size is essential since the training data will also suffer from the corruption. For large size data, a more complex structure with more layers and hidden units will be used. Training epoch and learning rate are chosen by compare the L2 training error and test error.

Moreover, since we are not able to get the ground truth in practice, test error can't be calculated to assist us tuning hyperparameters. The test error we mention above is actually a validation step, the "test" data is from the complete part of data and is manually corrupted.

3) Mean replacement plus Autoencoder

Training and test data pre-processing will the same as it in VAE

Algorithm:

1. Get the training data using the same method mentioned in VAE
2. Use the data above to train a multilayer autoencoder
3. Replace missing values in test data with mean
4. Input the test data to the trained AE and do prediction

Hyperparameters:

Architecture, optimizer, batch_size, learning rate, training epoch

Strategy for tuning hyperparameters:

AE and VAE share the same strategy

4) KNN

When features are missing, they can't contribute in distance calculation, which will cause a bias in nearest neighbor determination. To address this issue, a masked Euclidean distance is used here [2]. In particular, the missing data for a particular feature will be ignored and the weights for other features will be increased accordingly. The masked distance between two samples x_1 and x_2 will be:

$$Dist(x_1, x_2) = \frac{number\ of\ all\ features}{number\ of\ remaining\ features} * L2(x_{1_{remain}}, x_{2_{remain}}) \quad (6)$$

where $x_{1_{remain}}$ and $x_{2_{remain}}$ are the remaining features for x_1 and x_2 , L2 is L2-norm. Missingpy package is used for KNN imputation [9].

Algorithm:

1. Replace all missing data with np.nan
2. A masked Euclidean distance is computed between the current target and all other instances
3. Fill each missing position with the average of their K neighbours
4. If an instances has too many missing values, we will directly apply mean replacement algorithm described below.

Hyperparameters:
K(N_neighbors), weights for neighbors, metric.

Strategy for tuning hyperparameters:
Try different K values to find the one that has smaller l2 loss.

3. Experiments

a) Toy Dataset

Data generation:
The synthetic data set we use is the same as the data described in [8]. The dataset consists of four variables, parameterised by a single variable z .

$$z \sim U(0,1) \tag{7}$$

Further parameters are derived from z:

$$r = sqrt(z) \tag{8}$$

$$\eta = 0.25 \times \pi \times z \tag{9}$$

The four variables are then sampled from Gaussian distributions as follows:

Table 1. Synthetic data generation

Mean	Distribution
$x_{1,\mu} = r \times \cos\eta$	$x_1 = \mathcal{N}(\mu = x_{1,\mu}, \sigma^2 = 0.07 \times z)$
$x_{2,\mu} = 5r \times \sin\eta$	$x_2 = \mathcal{N}(\mu = x_{2,\mu}, \sigma^2 = 0.06 \times z)$
$x_{3,\mu} = 2r \times \tan\eta$	$x_3 = \mathcal{N}(\mu = x_{2,\mu}, \sigma^2 = 0.05 \times z)$
$x_{4,\mu} = 3 \times exp(-\eta)$	$x_4 = \mathcal{N}(\mu = x_{2,\mu}, \sigma^2 = 0.04 \times z)$

We know their density distribution and this synthetic dataset is a non-linear dataset which will help us to identify the good method not only work good on linear dataset but also good at non-linear data.

Missing values for *toy dataset*:
Missing data was introduced to the data using the same way as [8]. In particular, we will generate two kinds of datasets, a lightly corrupted dataset and a heavily corrupted dataset. The uncorrupted datasets were used to calculate the accuracy of the imputation methods.

The lightly corrupted dataset was created by randomly selecting 20% of the rows of a dataset, and replacing a randomly selected variable in each of these rows with NaN (not a number). Further missing values were introduced

by randomly selecting 50% of the rows of the lightly corrupted datasets, and replacing a randomly selected variable in each of these rows with NaN. This procedure was repeated three times, resulting in highly corrupted datasets.

The resulting two datasets have 95% and 60% of original data. The same method to generate missing value will also be applied in the following datasets.

b) Real-world Datasets

Milling circuit dataset

The milling circuit data is based on the simulation described in [10], with disturbances added in [11]. The milling circuit has three control loops, and the circuit responds to stochastic variations (representing disturbances) in the simulated ore size distribution and grindability, as well as incorporating measurement error. The original data from the simulation consists of samples every 30 seconds, for 104 days. The data has been subsampled to a sample every 15 minutes, in order to introduce some independence between samples (a key assumption in most data imputation approaches). This resulted in a set of 9985 data points.

Missing values for *milling circuit dataset*:

Generated in the same way as in toy dataset

MNIST dataset

MNIST dataset is a large database of handwritten digits, here we use the Keras to load the data [12].

Missing values for *MNIST dataset* :

A missing rate (0.3) will be used to decide the portation of missing pixels.

c) Experimental protocol

Evaluation methods

1. Plot RMSE for different methods and datasets
2. For MNIST dataset, we will also examine the quality of reconstructed images by visualization

Code resources

1. Some built-in model in Keras [13];
2. Tensorflow modules [14];
3. Missingpy package for KNN [9];
4. Variational autoencoder [15].

e) Results:

Toy dataset:

Table 2: The RMSE error for the synthetic dataset(error are calculated after data standardised*)

Methods	Lightly corrupted	Heavily corrupted
Baseline	0.997	0.995
VAE	**0.020**	**0.134**
KNN	0.103	0.255
AE	0.151	0.229

*Data is standardized before training and imputing to make the values have zero-mean and unit-variance

As we can see from the above table, all three strategies outperform baseline method. Mean-replacement plus VAE achieves lowest RMSE. KNN also did well, which verifies our assumption that it may be a good method on

small-size features dataset. However, KNN is not able to get the correct imputation when samples are heavily corrupted with a few know features.

Results for VAE:

The visualization of distributions of the ground truth and the imputed data for lightly corrupted dataset is shown The in Fig.1. Ground truth and imputed values are plot on the same image and distribution for each variable can be found in diagonal line. As it shows in Fig.1, VAE successfully reproduces the non-Gaussian distributions and nonlinear relationships between the four variables; similar results are achieved for both datasets, for both levels of corruption.

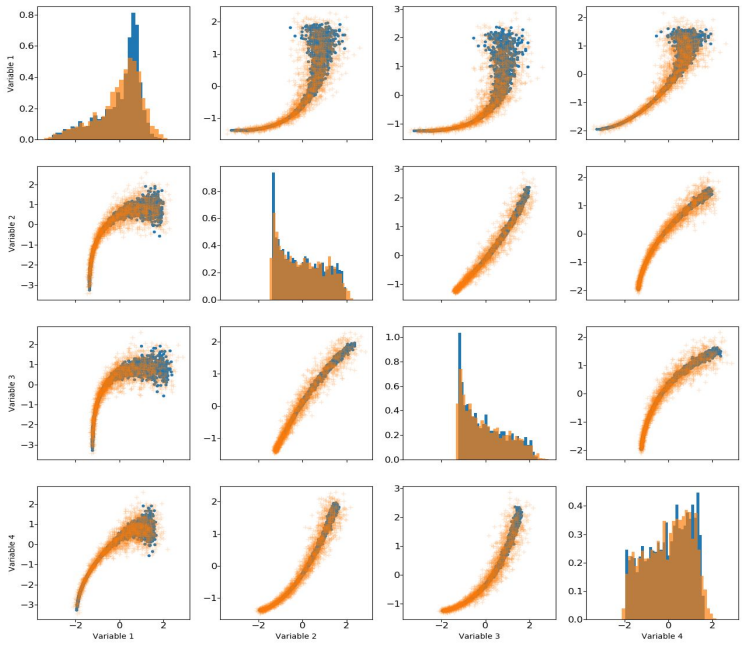
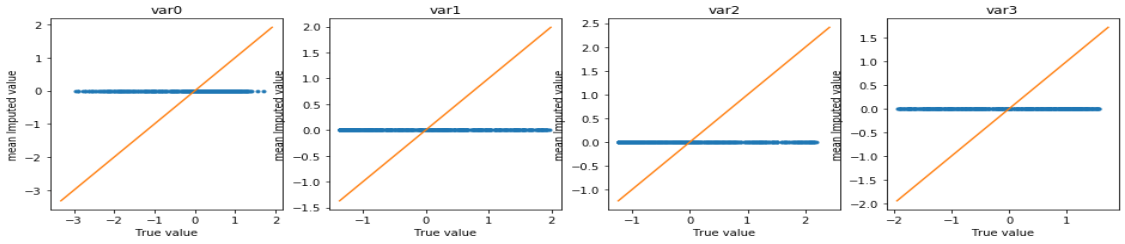


Figure 1. Comparison of lightly corrupted synthetic data (blue dots) with data generated by the VAE (orange plusses), after 200 training iterations. Variable histograms are shown on the diagonal, and scatter plots of variable pairs on the off-diagonals.

Performances for all methods:

VAE's imputation results are shown in Fig.2. The results are very similar for heavily corrupted data. Each row is one method we proposed above and each column is one feature of our dataset. When the blue dots are on the yellow line, it means that the truth value and the imputed value are the same. As shown in Fig.2, VAE has the largest overlapping area and thus is the best model for this dataset.



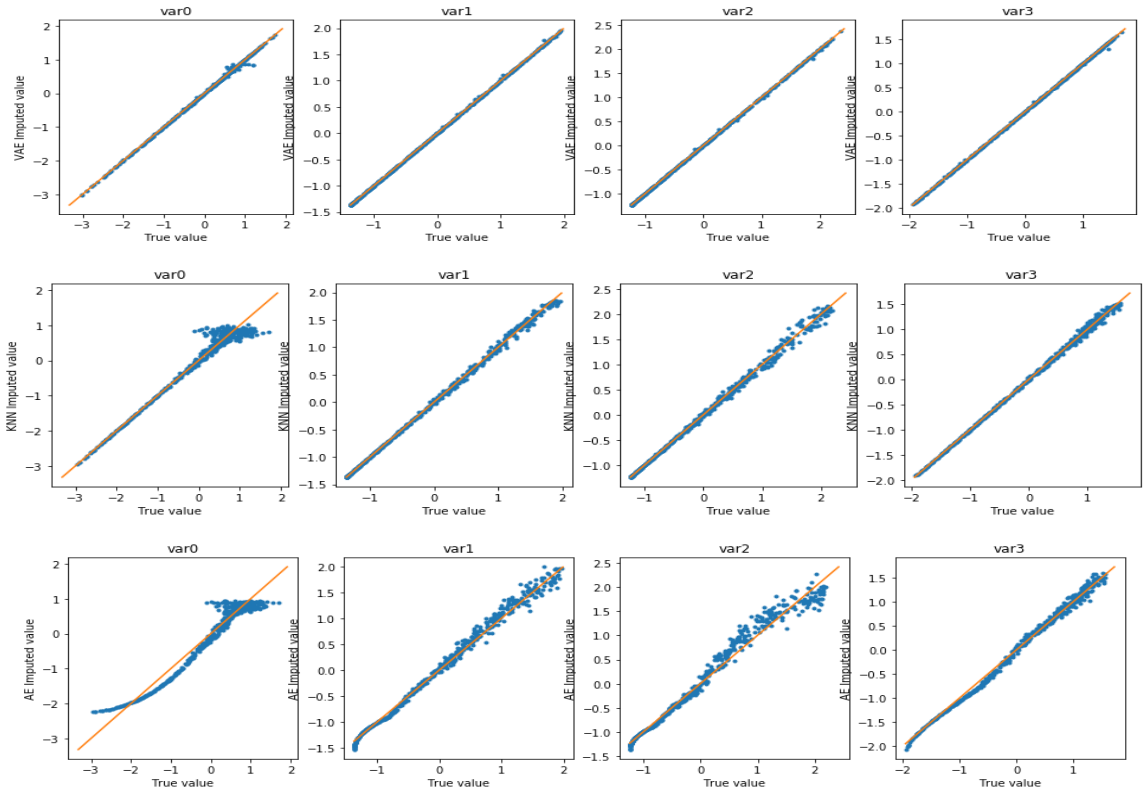


Figure 2. The visualization of distributions of the truth data and the imputed data for toy dataset. From top to down is baseline method, VAE method, KNN method and AE method.

VAE's uncertainty

One advantage of VAE method is it predict a distribution rather than an exact value. As shown in Fig. 3, we sampleed one instance and plot VAE's predictions as a Gaussian distribution using its mean and variance. The truth value and AE's prediction are plotted as red and blue horizontal lines respectively. The imputation results of VAE method is more likely to cover the truth value than that of AE.

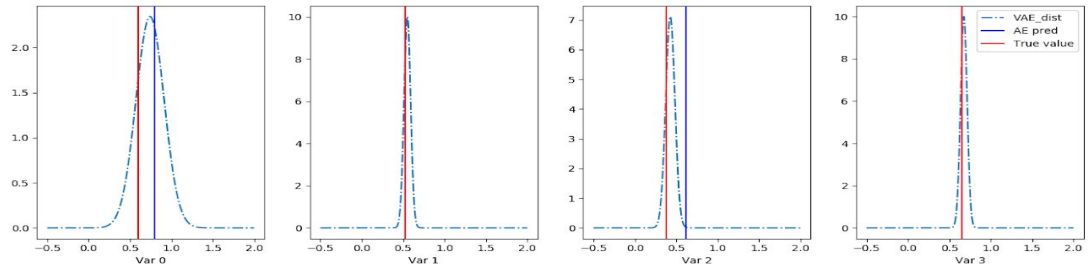


Figure 3. Comparison between AE and VAE imputation values. Each column stands for one feature.

MNIST:

Hyperparameters:

Because of the large size of the data, tuning hyperparameters becomes an essential part. For VAE model, we consider two components, lower bound and train-test error, by adjusting batch size, learning rate and latent space dimension, we find the best combination of hyperparameters, as shown in Fig.4 and Fig.5. Hyperparameter setting can be found in Appendix.

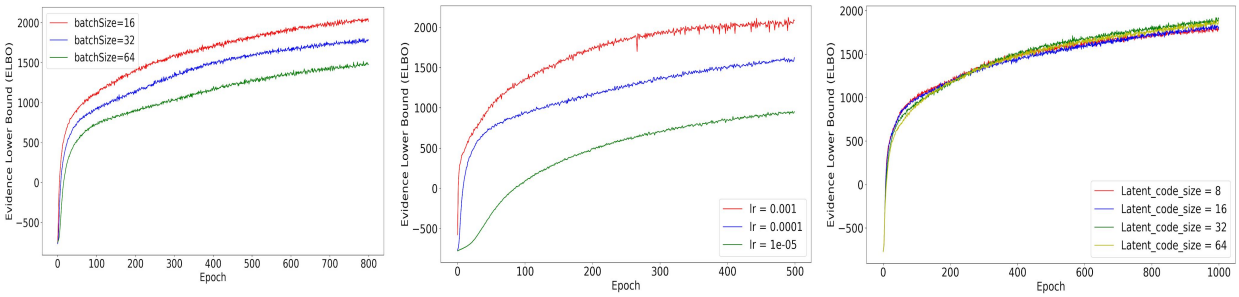


Figure 4. The Evidence lower bound. From left to right, the hyperparameters are batch size, learning rate and latent space dimension.

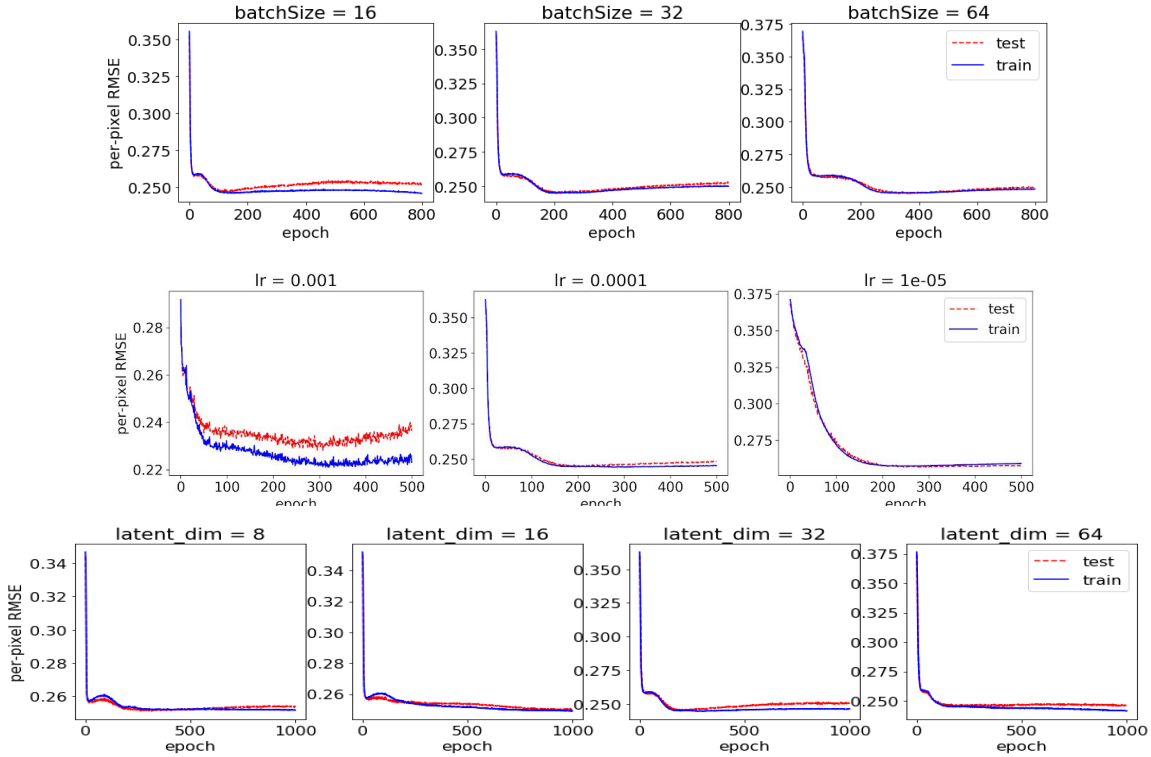
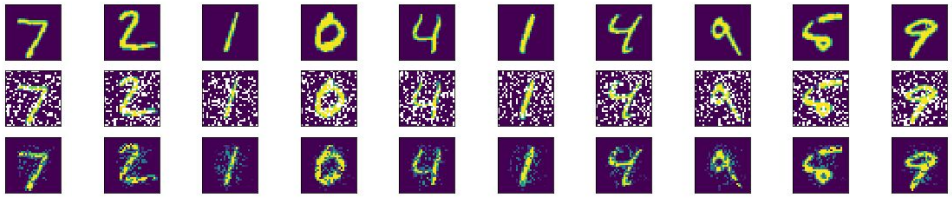


Figure 5. Test and training error. The hyperparameters are in the same order as in Fig.4. As epoch increases, all of them appears to overfit the training data.

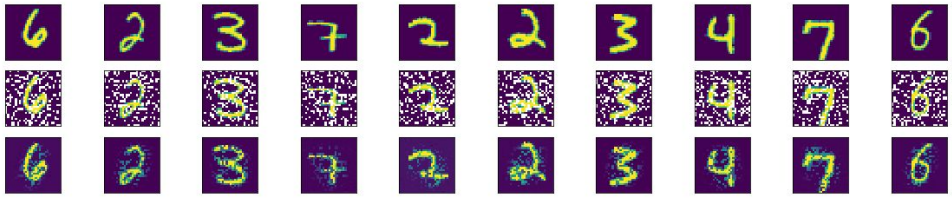
Results:

Using a 0.3 missing data rate, we generate some corrupted pictures and then impute the missing pixels using a trained model. The results are shown below (Fig.6), hyperparameter selection for KNN can be found in Appendix. We can see that most of figures can be inferred to its original class. Among them, VAE have the clearest results with smallest amount of noise.

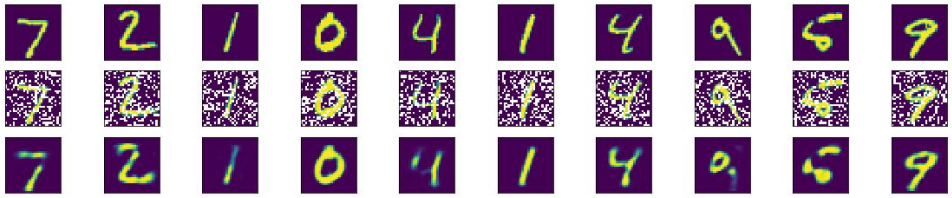
Mean



VAE



AE



KNN

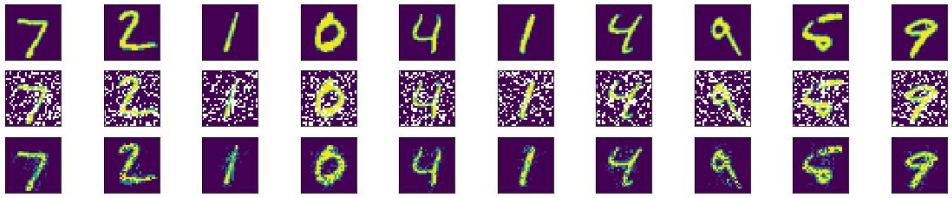


Figure 6. The visualization of the imputation results by baseline, VAE, AE and KNN method. The top row for each group is the original pictures. The middle row is corrupted pictures with a missing data rate of 0.3. The third row shows results after AE's imputation.

Milling circuit:

The imputation results are similar to the toy dataset. Please check Appendix for detail.

Conclusion:

We test four methods to impute missing data, among them, VAE performs the best on both small and large dataset. KNN is also a good choice for dataset with small feature size. Moreover, KNN works well for sparse data, as shown in Fig.6, KNN's performance is almost as good as VAE, this is because the digit figure of MNIST is a sparse vector, KNN can take advantage of this property. In contrast to the MNIST dataset, KNN did bad for Milling circuit (refer to Appendix). These results verify our hypothesis.

4. Reflection and outlook

Short-term future work:

The first thing we can improve is tuning hyperparameters for models on different datasets. We made a mistake that we use the same batch_size in VAE model for both lightly and heavily corrupted dataset, this leads to a bad imputation results, which is also shown in our presentation, a strange plateau for the first variable of the toy dataset. This remind us that our models haven't achieved their potential. After changing this parameter, VAE achieved very good imputation results for toy dataset and lightly corrupted milling circuit data, but we still need to evaluate the performance of VAE on heavily corrupted milling circuit data.

Long-term future work:

We only use real-value dataset to run experiments, the categorical dataset remains unexplored. This will also be an interesting problem to study. To extend our methods to discrete categorical data, we can use a one-hot vector to express an instance.

Another work is to further explore KNN model in data imputation. We found that KNN performs as good as VAE method on MNIST dataset, we think this is because the MNIST data is sparse, which means each picture has very distinct boundary between dark background and light digit. KNN method can thus take advantage of this sparsity. One way to prove this is doing an experiment using a denser dataset such as facial figures.

Take-home lessons on BDL method:

We realize that not all hyperparameters have the same effect on results, especially for VAE model. In this project, we find the batch size in VAE training has a large influence on the final performance. We tried different batch size for different datasets and got quite different results which is shown in Experiment Section.

Tuning hyperparameters for VAE is kind of a tough task since there are so many hyperparameters and it's heavily relies on experience. However, carefully analysis before doing experiments really will help a lot. By first thinking carefully about the data, we had some ideas about which hyperparameter may matters a lot. Then we verify our hypothesis by doing several experiments and finally pick out the hyperparameters which are also mentioned above. We think this is a valuable lesson.

5. References

[1] Roderick JA Little and Donald B Rubin. Statistical analysis with missing data. John Wiley & Sons, 2014.

[2] Cátia M. Salgado, Carlos Azevedo, Hugo Proença and Susana M. Vieira, Missing Data, Secondary Analysis of Electronic Health Records, 10.1007/978-3-319-43742-2_13, (143-162), (2016).

[3] Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein and Russ B. Altman, Missing value estimation methods for DNA microarrays, BIOINFORMATICS Vol. 17 no. 6, Pages 520-525 (2001). https://academic.oup.com/bioinformatics/article/17/6/520/272365

[4] Kingma, Diederik P and Welling, Max. Auto-Encoding Variational Bayes. In The 2nd International Conference on Learning Representations (ICLR), 2013. https://arxiv.org/abs/1312.6114

[5] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. In Proceedings of the 27th International Conference on Machine Learning, pages 3371–3408. ACM, 2010. https://dl.acm.org/citation.cfm?id=1953039

[6] Fulufhelo V Nelwamondo, Shakir Mohamed, and Tshilidzi Marwala. Missing data: A comparison of neural network and expectation maximisation techniques. arXiv preprint, arXiv:0704.3474, 2007. https://arxiv.org/abs/0704.3474

[7] D. Kingma and J. Ba. Adam: A method for stochastic optimization. ICLR, 2015 https://arxiv.org/abs/1412.6980

[8] McCoy, John T., Steve Kroon, and Lidia Auret. "Variational Autoencoders for Missing Data Imputation with Application to a Simulated Milling Circuit." IFAC-PapersOnLine 51.21 (2018): 141-146. https://www.sciencedirect.com/science/article/pii/S2405896318320949

[9] https://github.com/epsilon-machine/missingpy

[10] le Roux, J.D., Craig, I.K., Hulbert, D.G., Hinde, A.L., 2013. Analysis and validation of a run-of-mine ore grinding mill circuit model for process control. Miner. Eng. 43, 121–134. doi:10.1016/j.mineng.2012.10.009

[11] Wakefield, B.J., Lindner, B.S., McCoy, J.T., Auret, L., 2018. Monitoring of a simulated milling circuit: Fault diagnosis and economic impact. Miner. Eng. 120, 132–151. doi:10.1016/j.mineng.2018.02.007

[12] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86, 2278–2324. http://yann.lecun.com/exdb/mnist/

[13] https://keras.io/

[14] https://www.tensorflow.org/api_docs/python/

[15] https://github.com/ProcessMonitoringStellenboschUniversity/IFAC-VAE-Imputation

Appendix

Toy dataset

Hyperparameter setting for VAE:

Training epoch	1000 for light corrupted dataset 10000 for heavily corrupted dataset
Impute teration times	25
Hidden layers	5
Hidden units for each layer	8,16,4,16,8
dimensionality of latent space	4
batch_size	32

Hyperparameter setting for AE:

Training epoch	1000 for light corrupted dataset 10000 for heavily corrupted dataset
Impute teration times	5
Architecture	2 layers for both encoder and decoder
Hidden units for each layer	8,16,4,16,8
Batch_size	32

Hyperparameter setting for KNN:

n_neighbors	5
weights for neighbors	Uniform
Metric	Euclidean

MNIST Dataset

Optimized Hyperparameter setting for VAE:

Training epoch	200
Impute teration times	25
Architecture	2 hidden layers for both encoder and decoder
Hidden units for each layer	128, 64, 32
Batch_size	32

Hyperparameter setting for AE:

Training epoch	200
Impute teration times	25
Architecture	2 hidden layers for both encoder and decoder
Hidden units for each layer	128, 64, 32
Batch_size	32

KNN tuning hyperparameters for MNIST dataset:

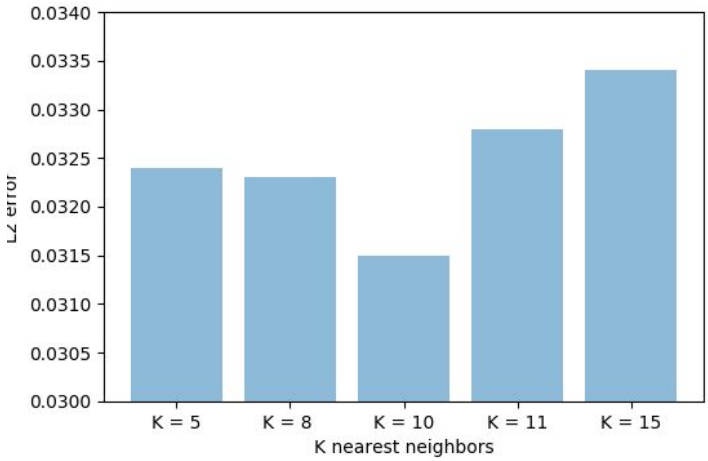
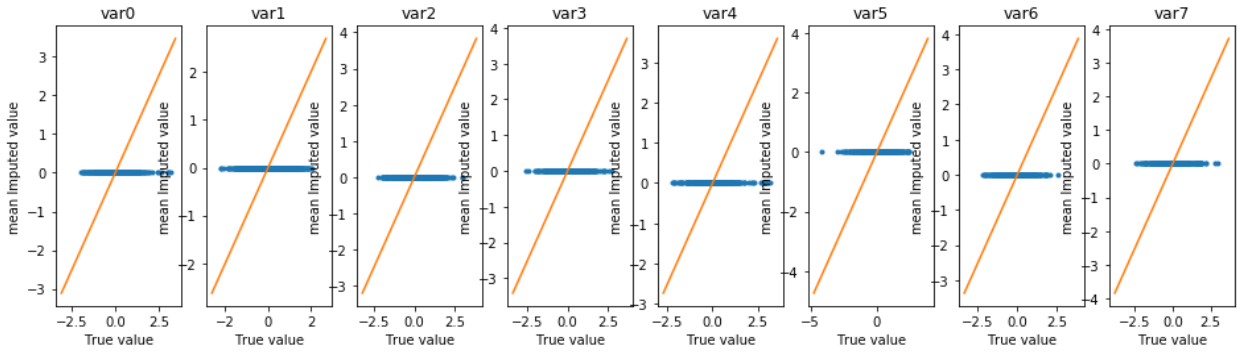


Figure 7. Hyperparameter tuning for KNN

For milling circuit dataset:

Hyperparameters are set the same as in toy dataset.

Results



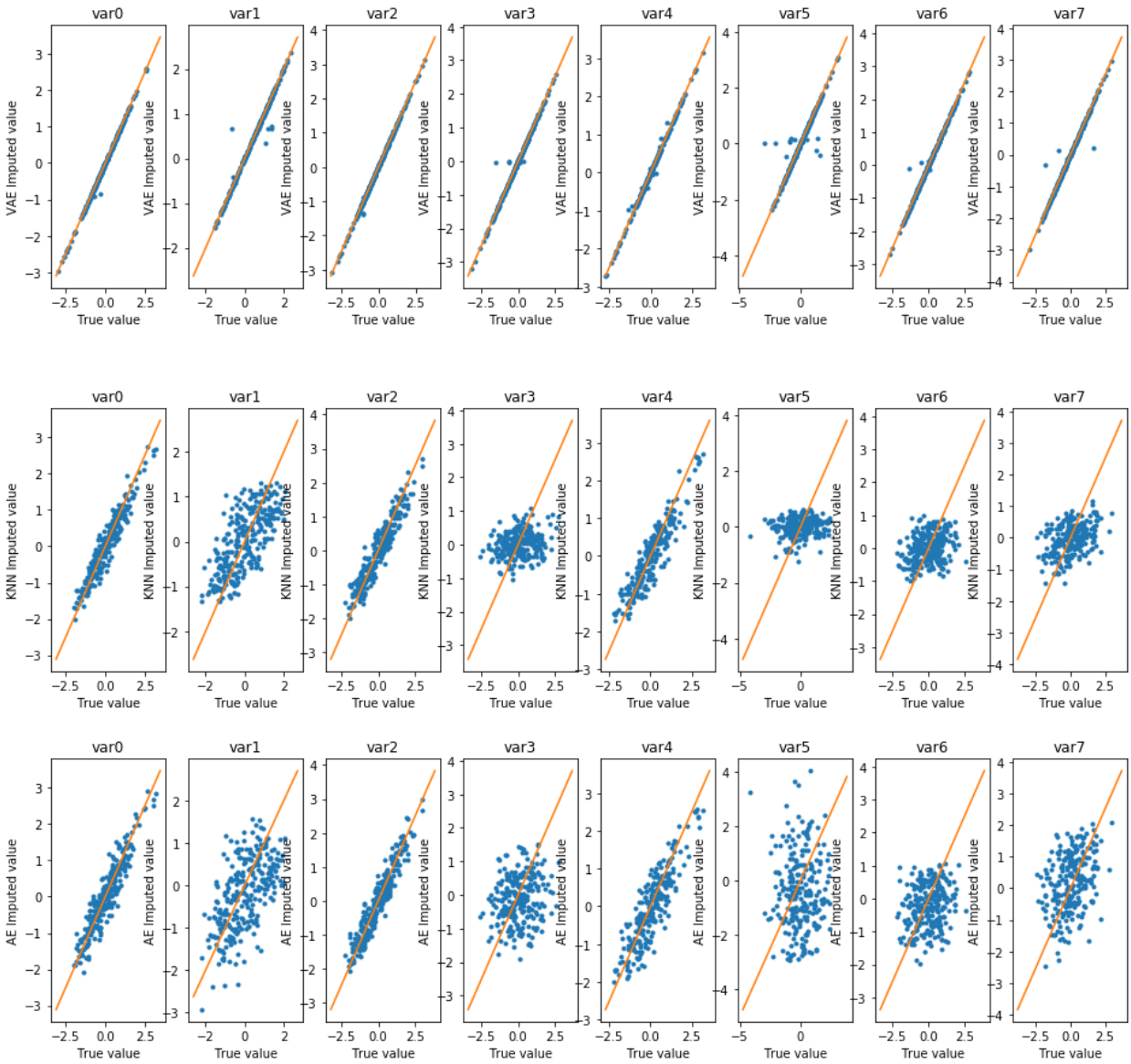


Figure 8. The visualization of distributions of the truth data and the imputed data for milling circuit dataset