ISSN 2303-4521
Periodicals of Engineering and Natural Sciences *Original Research*
Vol. 10, No. 3, June 2022, pp.350-367

Enhancing imputation techniques performance utilizing uncertainty aware predictors and adversarial learning

Wafaa Mustafa Hameed [1], Nzar A. Ali [2]
[1] Technical Collage of Informatics, Sulaimani Polytechnic University, Iraq
[2] Department of Statistics and informatics, University of Sulaimani, Iraq
[1,2] Department of Computer Science, Cihan University Sulaymaniya, Iraq

ABSTRACT

One crucial problem for applying machine learning algorithms to real-world datasets is missing data. The objective of data imputation is to fill the missing values in a dataset to resemble the completed dataset as accurately as possible. Many methods are proposed in the literature that mostly differs on the objective function and types of the variables considered. The performance of traditional machine learning methods is low when there is a nonlinear and complex relationship between features. Recently, deep learning methods are introduced to estimate data distribution and generate values for missing entries. However, these methods are originally developed for large datasets and custom data types such as image, video, and text. Thus, adopting these methods for small and structured datasets that are prevalent in real-world applications is not straightforward and often yields unsatisfactory results. Also, both types of methods do not consider uncertainty in the imputed data. We address these issues by developing a simple neural network-based architecture that works well with small and tabular datasets and utilizing a novel adversarial strategy to estimate the uncertainty of imputed data. The estimated uncertainty scores of features are then passed to the imputer module, and it fills the missing values by paying more attention to more reliable feature values. It results in an uncertainty-aware imputer with a promising performance. Extensive experiments conducted on some real-world datasets confirm that the proposed methods considerably outperform state-of-the-art imputers. Meanwhile, their execution time is not costly compared to peer state-of-the-art methods.

Keywords: Imputation Technique, Uncertainty Estimation, Adversarial Learning, Deep Neural Network.

Corresponding Author:

Wafaa Mustafa Hameed
Technical Collage of Informatics
Sulaimani polytechnic University
Sulaimani 64001, Iraq
wafaa.mustafa@spu.edu.iq

1. Introduction

Imputation is a technique that fills missing entries in any data source with appropriate values based on available information. The objective is to impute the missing values in the input dataset to resemble the underlying completed dataset as accurately as possible. This technique retains the number of samples in a dataset. By imputing all missing values in a dataset, we can analyze it using many standard methods developed for complete data. Missing values can arise from different sources such as measurement errors, low signal-to-noise ratio (SNR), or by deleting aberrant values. Thus, many datasets in the real world contain missing values which can be in any form such as NA, NAN, NULL, or blank. More formally, the problem can be defined as follows.

Let $\mathrm{X}^{(\mathrm{c})}$ be a random variable in d-dimensional space $\mathcal{X}^{(\mathrm{c})} = \mathcal{X}_1^{(\mathrm{c})} \times \mathcal{X}_2^{(\mathrm{c})} \times \dots \mathcal{X}_\mathrm{d}^{(\mathrm{c})}$. For each dimension j, we consider a new element $\mathrm{NA} \notin \mathcal{X}_\mathrm{j}$ that shows the missing value. Now, we define a new d-dimensional space $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \dots \mathcal{X}_\mathrm{d}$ where $\mathcal{X}_\mathrm{j} = \mathcal{X}_\mathrm{j}^{(\mathrm{c})} \cup \{\mathrm{NA}\}$. Also, let $\mathrm{M} \in \{0,1\}^\mathrm{d}$ be a binary random variable called mask that indicates which components of $\mathrm{X}^{(\mathrm{c})}$ are observed, that is $\mathrm{M}_\mathrm{j} = 0$ (resp. 1) if $\mathrm{X}_\mathrm{j}^{(\mathrm{c})}$ is revealed (resp. missing).

The input of an imputer algorithm is a dataset $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{m}_1), (\mathbf{x}_2, \mathbf{m}_2), \dots, (\mathbf{x}_\mathrm{n}, \mathbf{m}_\mathrm{n})\}$ where $\mathbf{x}_\mathrm{i} \in \mathcal{X}$ and $\mathbf{m}_\mathrm{i} \in \{0,1\}^\mathrm{d}$ is a mask corresponding to $\mathbf{x}_\mathrm{i}$. The aim of an imputer is to fill the unobserved entries in each $\mathbf{x}_\mathrm{i}$ with plausible values. In ref. [1] it was classified three missing mechanisms based on independency relation between observed and missing values:

MCAR: MCAR (Missing Completely at Random), in the missing value of a feature neither depends on observed data nor missed data. More precisely, $\mathrm{p}\left(\mathrm{m}|\mathrm{x}^{(\mathrm{c})}\right) = \mathrm{p}(\mathrm{m})$.

MAR (Missing at Random): where a missing value only depends on observed values but is independent of missing data. More precisely, $\mathrm{p}\left(\mathbf{m}|\mathbf{x}^{(\mathrm{c})}\right) = \mathrm{p}\left(\mathbf{m}|\mathbf{x}^{(\mathbf{obs})}\right)$ where $\mathbf{x}^{(\mathbf{obs})}$ indicates observed values of $\mathbf{x}$.

NMAR (Not Missing at Random): in which a missing value depends on other missing values and possibly on observed values. Dropping rows or columns in a dataset containing missing values comes at the expense of losing valuable data. As an illustrative example [2], let $\mathbf{X} \in \mathbb{R}^{\mathbf{n}\times\mathbf{d}}$ be the input data in which each element is missing at random a probability equal to 1%. In the case of $\mathbf{d} = \mathbf{5}$, we expect 95% ($\approx \mathbf{0.99^5}$) of rows have no missing value. However, when $\mathbf{d} = \mathbf{300}$, only 5% of rows are complete. Therefore, imputing admissible values for missing places is necessary especially in a high-dimensional dataset. Many methods are proposed to impute missing data that mostly differ on the objective function, types of the variables considered (numerical, categorical, both), and assumptions about data distribution or missing data mechanism [3]. Some popular strategies are in the literature include univariate statistical, low-rank approximation [4], nearest neighbor search [5, 6], multiple imputations [7], probabilistic [8], and deep learning-based methods. Deep learning methods attempt to estimate data distribution and generate values for missing entries to preserve the joint and marginal distribution. They are typically based on variational autoencoders [9, 10], or Generative Adversarial Networks (GANs) [11, 12,13]. These methods are originally developed for large datasets and custom data types such as image, video, and text. Thus, adopting these methods for small and tabular datasets that are prevalent in imputation applications is not straightforward and often yields unsatisfactory results. For example, results reported in [14] showed that these methods did not achieve competitive performance on a variety of UCI datasets. On the other hand, while traditional machine learning methods often work well for small datasets and can handle any feature type, their performance is low when there is a nonlinear and complex relationship between features. Besides, by utilizing advanced regularization techniques from the deep learning domain such as adversarial neural networks, we can boost the performance of these algorithms, especially in an out-of-sample setting.

Most existing methods do not account for uncertainty in the input data. For example, at each iteration of the popular MICE method, a predictor is used to impute values for the current feature j based on other feature values. Here, some of these values are real while others are the outputs of previously learned imputers and so are uncertain. Recently, [15] developed an imputer for time-series data that considered the uncertainty in imputed values. However, the model developed based on variational autoencoders (VAE) and specially designed for large time-series datasets and has high complexity. To address these issues, we develop a simple neural network-based architecture that works well with small datasets and utilizes a novel adversarial strategy to estimate the uncertainty of imputed data. The uncertainty scores of features are passed to the imputer, and it fills the missing values by paying more attention to confident feature values. In addition to instance-based feature weighting, we elaborate a novel loss function that enforces the imputer to generate values so that for any target feature: 1) it can estimate values for known values in the feature precisely, 2) the predicted values for the missing entries obey the underlying data distribution and could confuse the adversarial neural network module so that it cannot distinguishes imputed values from real ones. In summary, the contributions of the proposed model are as followings:

1- We develop uncertainty-aware imputation methods by exploiting a novel adversarial strategy and the proposed hybrid loss function. Specially, we propose a new and simple strategy for training the adversarial module for the imputation task that outperforms the common training strategy used in the deep learning domain.

2- The proposed methods considerably outperform state-of-the-art methods on most evaluated datasets. Meanwhile, its training time is not costly compared to deep learning-based methods.

3- The proposed model has a very simple structure, can work with any feature type and small data. Also, the model can be trained in an end-to-end paradigm using any available neural network optimizers through Back-Propagation (BP).

Table 1 summarizes the main notations used throughout this paper. The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 presents the proposed model in detail. Implementation details of the model are provided in Section 4. Section 5 reports the experimental results and provide analysis along with comparison with the peer state-of-the-art methods. Finally, Section 6 concludes with Conclusion and Future Work.

2. Related work

Many strategies are developed for handling missing data. In this section, we classify and discuss these techniques with a focus on the works closely related to the proposed model.

Table 1. Summary of the main notations

Notation	Description
$\boldsymbol{X} \in \mathbb{R}^{n\times d}$	The input dataset containing missing elements
$\boldsymbol{X}^{(.)} \in \mathbb{R}^{b\times d}$	A mini-batch sampled from dataset
$\boldsymbol{X}_{:j}^{(.)} \in \mathbb{R}^{b}$	Column j of the matrix $\boldsymbol{X}^{(.)}$
$\boldsymbol{X}_{-j}^{(.)} \in \mathbb{R}^{b\times d-1}$	Matrix formed by dropping column j from $\boldsymbol{X}^{(.)}$
NA	Missing value
S_{NA}	The set of all columns having missing value
$\boldsymbol{C} \in [0,1]^{n\times d}$	Certainty matrix
$I(.;\boldsymbol{\theta}_j)$	Imputer j parametrized by $\boldsymbol{\theta}_j$
$D(.;\boldsymbol{\phi}_j)$	Discriminator j parametrized by $\boldsymbol{\phi}_j$

Univariate imputation fills a lost value in a feature only based on other values in that feature. They often impute a missing value by the mean, median, or mode of the corresponding attribute. These methods have a low computational cost, but they ignore the correlation among features and thus often lead to poor results. However, they are widely used as an initializer in many advanced techniques. While, In multivariate imputation, observed values of other features can be used to impute unknown entries in a variable. kNN is a multivariate imputation technique that for each instance with missing values finds k nearest neighbors in the training set. Then, it imputes a value for a missing entry based on values in the nearest neighbors. Some extensions include sequential kNN [6] and iterative kNN [5, 17]. In iterative imputation, each feature with missing values is considered as a function of other attributes. Let j be a selected column containing missing values. Then, we treated other features in the dataset as inputs X_{-j}, and fit a regressor on (X_{-j}, j) for any known values in j. Subsequently, the trained regressor is utilized to predict the unknown values of j. A common regressor used in this approach is the least squares [18]. Some work also explored other regressor types such as *Support Vector Regression* (SVR) [19]. Probabilistic approaches assume parametric joint distribution on the entire dataset. The *Expectation Maximization* (EM) is utilized to estimate model parameters and missing values by maximizing the log likelihood function [9]. These approaches provide good theoretical properties but lack flexibility aspects. For example, in the case that the input dataset contains both numerical and categorical features, multivariate distributions often failed to model the underlying data distribution. Linear regression-based methods may have a poor performance when a nonlinear relationship exists among variables. To address this issue, regression trees are employed for imputation in [20]. Moreover, [21] extends random forest for the imputation task and obtains promising results. Tree based methods are non-parametric and do not consider any specific distribution of data. Predictive Mean Matching (PMM) is a popular traditional imputation method. Let j be a variable with some missing values and $\mathrm{X_{-j}}$ denote a set of variables with no missing entries. PMM utilizes logistic regression to assign an appropriate weight (w) to each variable in $\mathrm{X_{-j}}$. These weights define the posterior predictive distribution $\mathrm{p(j|X_{-j})}$. Then, it samples a new set of weights $\mathbf{w}^*$ from the distribution and predicts values of j for all cases in the dataset. Let I (resp. K) be the set of instances with missing values of j (resp. no missing). For each instance i in I, PMM finds a set of cases in K that their predicted values are close to the predicted value of i. Then, it imputes the value of i by randomly choosing a value from this set. (Singular Value Decomposition) SVD imputation [22] assumes input data are noisy observations produced by linear combinations of a small set of principal components. SVD learns these components from the dataset and then imputes the missing entries from a linear combination of them in an iterative process. Bayesian principal component imputation [23] extends the SVD method to incorporate information about prior distribution on the model parameters.

Multiple imputation creates multiple copies of the dataset and estimates values for missing entries for each dataset. The imputed outcomes are then combined using an appropriate strategy. These methods often have three following steps [22]:

1- Imputation: that is like to single imputation, however, the imputed values come from m datasets rather than just one dataset.
2- Analysis: each of m imputed values is analyzed.
3- Pooling: the results are combined by calculating the mean, variance, and or any other combining technique.

MICE (multiple imputation by chained equations) [7]: is a well-known multiple imputation technique. It assumes the value of missing data depends on the observed data. It generates a series of regressors or any other models to impute multiple values for a missing entry. First, a simple mean imputation is applied to each missing entry referred as placeholder. Second, the "placeholder" mean imputations for one feature are set back to missing. Third, a suitable regressor is fitted to impute values for the missing variable. These steps are repeated until the max iteration defined by the user reaches. Once the max number of iterations is completed, the entire process is repeated to generate the next completed dataset. The following flowchart illustrates the main steps in the MICE approach. MICE can use PMM, logistic regression, Bayesian linear regression, and similar methods as the regressor.

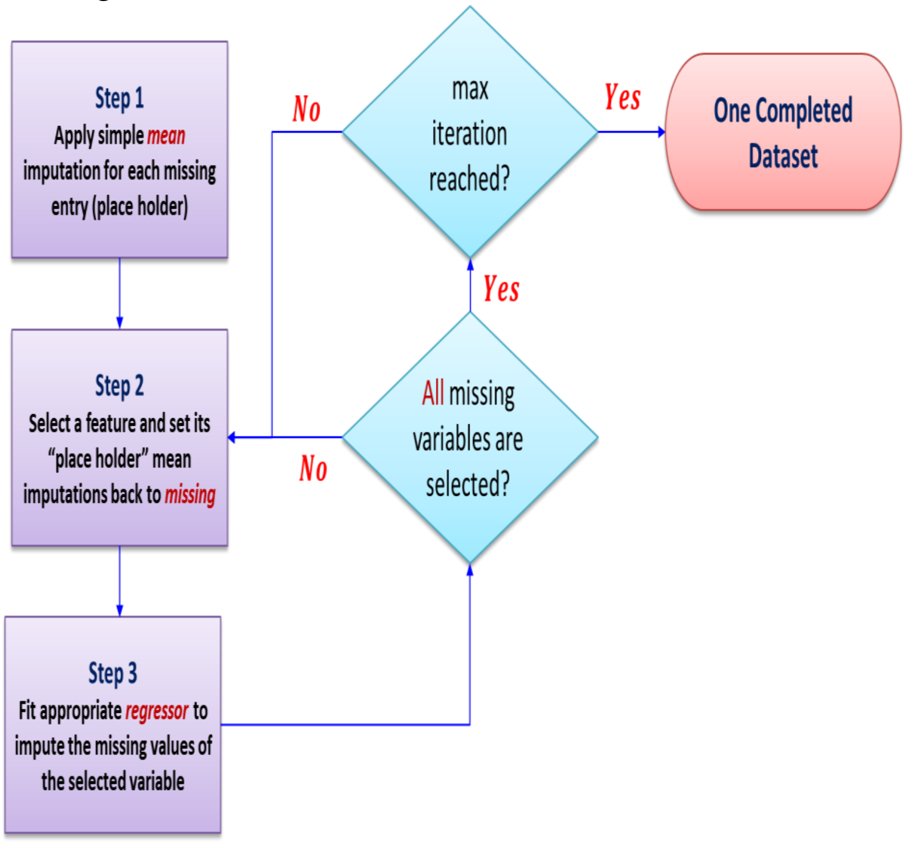

Step 1
Apply simple mean imputation for each missing entry (place holder)
Step 2
Select a feature and set its "place holder" mean imputations back to missing
Step 3
Fit appropriate regressor to impute the missing values of the selected variable
All missing variables are selected?
No
Yes
max iteration reached?
No
Yes
One Completed Dataset

Figure 1. Flowchart of MICE

The Amelia method is also a multi-imputation method based on EM and bootstrapping. It assumes data are drawn from a multivariate Gaussian [25]. [10] analyses the imputation problem from the optimization perspective and develops optimization-based methods for both single and multiple imputation that refines existing methods in the out-of-sample setting.

2.1. Deep Learning-Based Imputation

Deep learning-based approaches for data imputation are often generative and developed by extending auto-encoders and GAN models for incomplete data. MIDA (Multiple Imputation using Denoising Autoencoders) [10] uses an overcomplete Denoising Autoencoder (DAE) for data imputation. The model projects the input data to a higher dimensional space to recover missing information.
Figure *2* depicts the architecture where each layer in the encoder increases the dimensionality by adding Θ neurons. In the decoding stage, it sets half of the input neurons to zero and aim to recover the complete data.

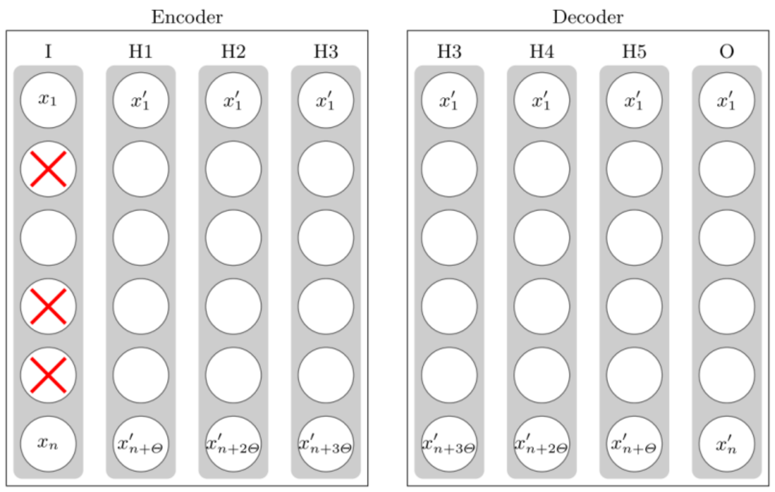


Figure 2. The architecture of encoder and decoder parts of MIDA [11]

The main problem is that autoencoders require complete data for training. Therefore, MIDA initially uses mean imputation to fill missing entries producing fake complete data. Also, by this approach, it is impossible to learn missing patterns in the original dataset. Generative Adversarial Imputation Nets (GAIN) [13] extends the GAN model to impute missing values. Like standard GAN, it has two components: the discriminator and generator. The generator aims to accurately fill the missing values whereas the discriminator tries to distinguish between imputed and real values. Ambient GAN [14] works only with image data type and extends the standard GAN to incorporate a measurement process such as adding noise, data removal, and projection. Here, the discriminator goal is to distinguish between real measurements from simulated ones. This approach assumes that the measurement process is known and has only a few parameters. However, these assumptions do not hold for many datasets with missing values. In addition to using a GAN (G_x, D_x) to learn data distribution, MisGAN [12] utilizes an auxiliary GAN (G_m, D_m) to learn mask distribution. Let $\boldsymbol{f}_\tau(\boldsymbol{x}, \boldsymbol{m})$ be a function that fills the missing entries in $\boldsymbol{x}$ by a constant τ. Also, let $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{m}}$ be the output of G_x and G_m respectively. The complete data generator G_x is trained so that its outputs when masked by $\tilde{\boldsymbol{m}}$ and $\boldsymbol{f}_\tau$ (i.e., $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{m}})$) can not be distinguished from the real partially observed data $(\boldsymbol{x}, \boldsymbol{m})$. Figure 3 illustrates the overall architecture of the MisGAN.

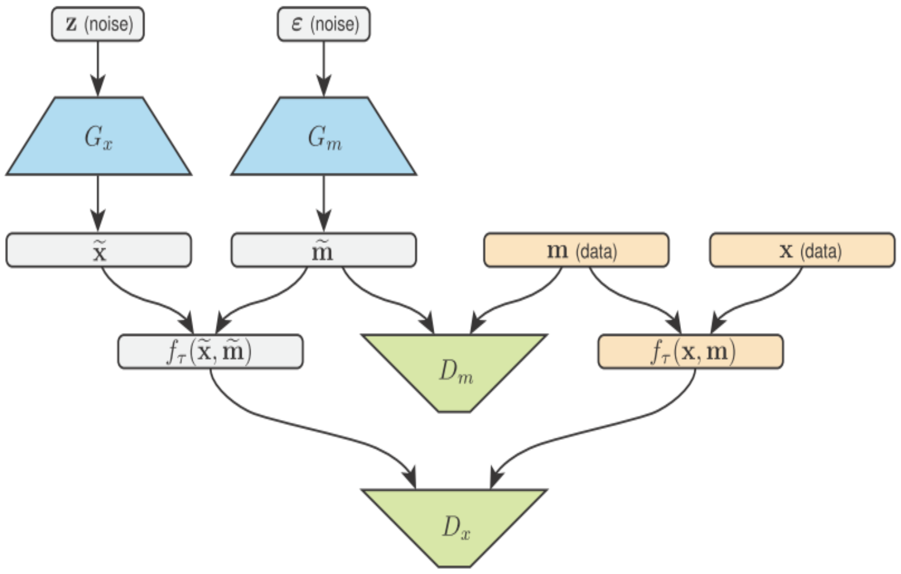


Figure 3. Architecture of the MisGAN

In[15], it is built on the assumption that two random batches from the same dataset should have the same distribution. It leverages the Sinkhorn divergence to measure the transport distance between two random batches and impute the missing values by minimizing this loss function.

3. The proposed method

The goal of the proposed model is to impute the missing values so that the adversarial neural network cannot distinguish real values from imputed ones. Also, we aim to account for the uncertainty of imputed values using confidence scores obtained from our adversarial module. Figure 4 illustrates the overall architecture of the proposed model.

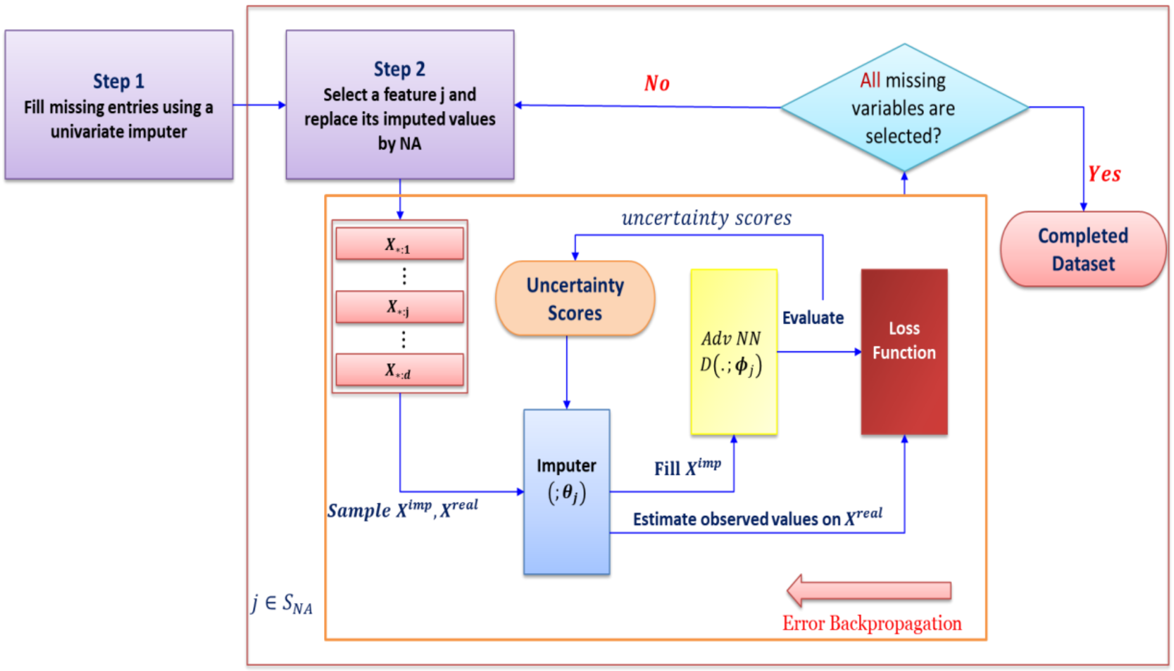


Figure 4. Overall architecture and training process of the proposed imputation model

In the first step, the missing places are filled using a univariate imputation technique like *mean*. Also, we consider a global certainty matrix $\boldsymbol{C} \in [0,1]^{n\times d}$ and initialize it using the mask matrix $\boldsymbol{M} = [\boldsymbol{m_1}, \boldsymbol{m_2}, \dots, \boldsymbol{m_d}]$ as $\boldsymbol{C} = \boldsymbol{M}$. Thus, in the first stage, we are certain only about real (observed) values in the dataset. We choose a feature at each step according to an appropriate heuristic function. For example, the feature can be selected according to the increasing order of the number of missing entries. Let $j \in S_{NA}$ be the selected feature in the current cycle where S_{NA} denotes the set of all columns containing missing values. The variable j denoted by $\boldsymbol{X}_{:j}$ is considered as the target and other features denoted by $\boldsymbol{X}_{-j}$ form the input. To impute the missing values in the j^{th} column, we design a specific *imputer* $I(.;\boldsymbol{\theta}_j)$ parametrized by $\boldsymbol{\theta}_j$, and its corresponding *discriminator* $D(.;\boldsymbol{\phi}_j)$ parameterized by $\boldsymbol{\phi}_j$. Note that for each instance $\boldsymbol{x}_i$, some of its features have real values and so are confident, whereas some of them are filled using the corresponding imputer in the previous steps. There exists uncertainty about the imputed values that must be considered to build a more accurate imputer. More precisely, to optimize the imputer j, we aim to have more attention to high confident features per instance. Thus, we target weighting the features considering their certainty scores. The less confident features of an instance should get lower weights. The certainty score of an imputed value is obtained from the adversarial module as described in the following. The adversarial module aims to discriminate imputed values from real ones. We design the loss function to learn $\boldsymbol{\theta}_j$ such that the resulting imputer in addition to estimating a missing entry with high accuracy, be able to confuse the adversarial module so that:

1- It cannot distinguish between real and imputed values.
2- The imputed values have a similar distribution to the real values.

It implies a minimax game between the imputer and the adversarial module. On one hand, the imputer tries to fool the adversarial network, and on the other hand, the adversarial module attempts to discover discriminative information in the observed values and identifies fake values. In the following, we discuss about each component of the proposed model with more details. The adversarial module aims to estimate how much *the imputed values are similar to real* ones. We can model this module by a collection of adversarial neural networks $\{D(.;\boldsymbol{\phi}_j): \boldsymbol{x} \in \mathbb{R}^d \to [0,1]\}_{j\in S_{NA}}$ where the output $D_j(\boldsymbol{x};\boldsymbol{\phi_j})$ shows the probability that how much the input $\boldsymbol{x}$ is real. The standard training process in domain adaptation and GAN is to create mini-batches containing both real and imputed values of the target feature and then ask the discriminator D_j to distinguish real ones from imputed. However, we experimentally find out this process is not effective for small tabular datasets containing missing values. Therefore, we propose the following simple yet effective strategy. First, we sample $\mathbf{X}^{\text{real}}$ from instances having real values on feature j. Second, we corrupt the values on the target feature by adding some noises to the actual values and thus obtaining a fake sample $\mathbf{X}^{\text{fake}}$. The real and fake samples are mixed and passed to the adversarial module. Finally, we train the discriminator D_j to distinguish real from fake data accurately. A well-trained discriminator D_j acts like a critic that can judge how much a sample of imputed data is similar to a real one. To this end, we use the following BCE loss to train D_j:

$$\mathcal{L}_{dis}(\boldsymbol{X}^{real}, \boldsymbol{X}^{fake}) = \text{BCE}(\mathbf{1}, \boldsymbol{p}^{(j)}) + \text{BCE}(\mathbf{0}, \boldsymbol{p}^{(j)}) = \frac{1}{b}\sum_{i=1}^{b} -\log \boldsymbol{p}_i^{(j)} - \log(1 - \boldsymbol{q}_i^{(j)}), \tag{1}$$

where $b = |\boldsymbol{X}^{real}|$, $\boldsymbol{p}_i^{(j)} = D_j\left(\boldsymbol{x}_i^{(real)}; \boldsymbol{\phi}_j\right)$, and $\boldsymbol{q}_i^{(j)} = D_j\left(\boldsymbol{x}_i^{(fake)}; \boldsymbol{\phi}_j\right)$.

Figure 5 illustrates the overall training process of the adversarial module in the proposed model.

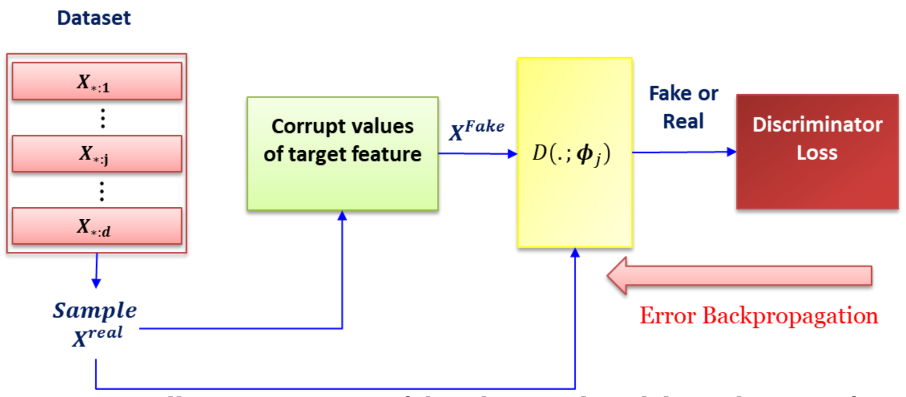


Figure 5. Overall training process of the adversarial module on the target feature

The imputer receives the weighted features as input and optimizes $\boldsymbol{\theta}_j$ according to the target feature j. For a numeric target feature, we model the imputer using a regressor neural network, and in the case that the target attribute is categorical, we employ a classifier neural network as the imputer. Nevertheless, the imputer module is not limited to a neural network, and one can implement it utilizing other gradient-based machine learning models. Figure 6 illustrates the overall training process of the imputer in the proposed model.

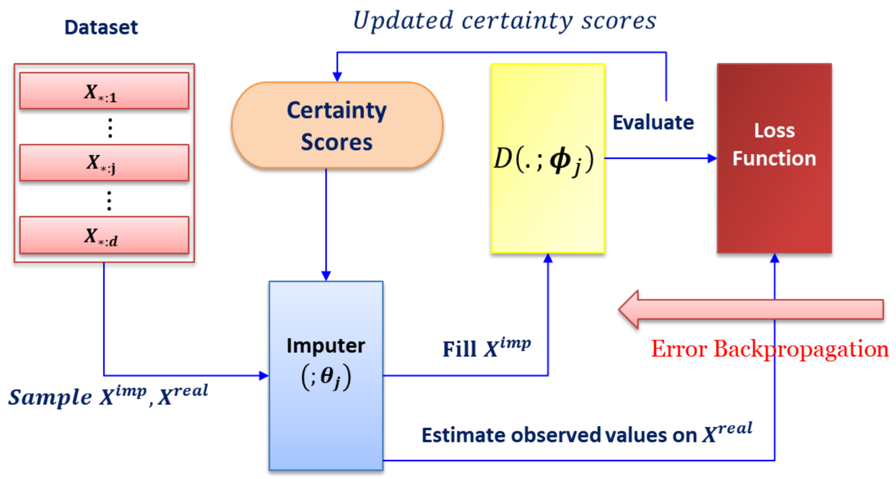


Figure 6. Overall training process of the proposed imputer on the target feature j

To optimize model j, we sample two batches from data: $\boldsymbol{X}^{real}$ and $\boldsymbol{X}^{imp}$ where $\boldsymbol{X}^{real}$ is sampled from data having real values on feature j whereas $\boldsymbol{X}^{imp}$ is chosen from data having missing values. We train the imputer j so that it can predict the target values $\boldsymbol{X}_{:j}^{real}$ accurately. On the other hand, it should fill missing entries on $\boldsymbol{X}^{imp}$ so that the adversarial module cannot distinguish the imputed values from real ones.
The proposed hybrid loss function is composed of two terms: 1) $\mathcal{L}_{imp}$: which ensures the imputed values follow the true distribution of data so that the adversarial module cannot distinguish between real and unobserved

entries, 2) $\mathcal{L}_{fit}$: which enforces the imputers to predict the missing entries with high accuracy. For a batch $\boldsymbol{X}^{imp} \in \mathbb{R}^{b\times d}$ sampled from instances having missing values on the feature j, first, imputer j fills the missing elements in column j of $\boldsymbol{X}^{imp}$. To this end, we mix the confidence scores with the input $\boldsymbol{X}^{imp}_{-j}$ using element-wise multiplication:

$$\boldsymbol{X}^{ic}_{-j} = \boldsymbol{X}^{imp}_{-j} \odot \boldsymbol{C}^{imp}_{-j}, \tag{2}$$

Then, we run $I(.;\boldsymbol{\theta}_j)$ on $\boldsymbol{X}^{ic}_{-j}$ and store the outputs in $\boldsymbol{X}^{imp}_{:j}$. Afterward, we pass $\boldsymbol{X}^{imp}$ to the adversarial module and get its output denoted by $\boldsymbol{p}^{(j)} \in \mathbb{R}^{b}$. The $\boldsymbol{p}^{(j)}_i$ approximates the probability that $\boldsymbol{x}^{imp}_i$ (i^{th} instance in $\boldsymbol{X}^{imp}$) is real. A well-trained imputer j should fill the missing elements in $\boldsymbol{X}^{imp}$ so that the adversarial module cannot classify any instance $\boldsymbol{x}^{imp}_i$as fake and its output should be near 1. Thus, we define $\mathcal{L}_{imp}$ as the Binary Cross-Entropy (BCE) loss between the target vector $\mathbf{1} \in \mathbb{R}^{b}$ and the predicted values $\boldsymbol{p}^{(j)}$:

$$\mathcal{L}_{imp}(\boldsymbol{X}^{imp}) = \mathrm{BCE}(\mathbf{1},\boldsymbol{p}^{(j)}) = \frac{1}{b}\sum_{i=1}^{b} -\log \boldsymbol{p}^{(j)}_i \text{ where } b = |\boldsymbol{X}^{imp}| \tag{3}$$

Besides, we update the j^{th} column of the certainty matrix $\boldsymbol{C}$ by setting its corresponding elements equal to $\boldsymbol{p}^{(j)}$:

$$\boldsymbol{C}[I^{(imp)}, j] = \boldsymbol{p}^{(j)}, \tag{4}$$

where $I^{(imp)}$ is the set of indices in original dataset corresponding to $\boldsymbol{X}^{imp}$.

For a real batch $\boldsymbol{X}^{real} \in \mathbb{R}^{b\times d}$, let $\boldsymbol{C}^{real} \in \mathbb{R}^{b\times d}$be the corresponding confidence scores obtained by the adversarial module. We combine the certainty scores with the input $\boldsymbol{X}^{real}_{-j}$ using element-wise multiplication:

$$\boldsymbol{X}^{rc}_{-j} = \boldsymbol{X}^{real}_{-j} \odot \boldsymbol{C}^{real}_{-j} \tag{5}$$

Now, let $\boldsymbol{t}^{(j)}$=$\boldsymbol{X}^{real}_{:j}$ be the real values and $\boldsymbol{p}^{(j)}$ show the output of the regressor when applied on $\boldsymbol{X}^{rc}_{-j}$. Note that we know the real values on the target feature j for each instance in $\boldsymbol{X}^{real}$. In the case that the attribute j is a continues number, we employ an appropriate regression loss term like *smooth-L1* to achieve high accurate predictions. It is defined as:

$$\mathcal{L}_{fit}(\boldsymbol{X}^{real}) = \frac{1}{b}\sum_{i=1}^{b} \mathcal{L}_i(\boldsymbol{t}^{(j)},\boldsymbol{p}^{(j)}) \text{ where } b = |\boldsymbol{X}^{real}| \text{ and}$$

$$\mathcal{L}_i(\boldsymbol{t}^{(j)},\boldsymbol{p}^{(j)}) = \begin{cases} 0.5\left(p^{(j)}_i - t^{(j)}_i\right)^2/\beta, & \text{if } \left|p^{(j)}_i - t^{(j)}_i\right| < \beta \\ \left|p^{(j)}_i - t^{(j)}_i\right| - 0.5\beta, & \text{otherwise} \end{cases}. \tag{6}$$

For a categorical target feature j with K distinct values, we develop the imputer j as a classifier that outputs $\boldsymbol{p}^{(j)} \in \mathbb{R}^{b\times K}$ for the input $\boldsymbol{X}^{rc}_{-j}$. Also, we convert $\boldsymbol{t}^{(j)}$ to one-hot encoding format and define $\mathcal{L}_i$ as a classification loss like cross-entropy:

$$\mathcal{L}_i(\boldsymbol{p}^{(j)},\boldsymbol{t}^{(j)}) = -\sum_{k=1}^{K} t^{(j)}_{ik} \log \mathbf{Softmax}\left(p^{(j)}_{ik}\right). \tag{7}$$

The final loss is formulated as:

$$\mathcal{L}(\boldsymbol{X}^{real},\boldsymbol{X}^{imp}) = \mathcal{L}_{fit}(\boldsymbol{X}^{real}) + \lambda\mathcal{L}_{imp}(\boldsymbol{X}^{imp}), \tag{8}$$

where the hyper-parameter λ balances the trade-off between the $\mathcal{L}_{fit}$ and $\mathcal{L}_{imp}$ losses.
Algorithm 1 summarizes the main steps of the proposed algorithm named *Uncertainty Aware Adversarial Imputer* (UA-Adv Imputer).

4. Implementation details

We implemented the model using *PyTorch* deep learning library. The features in input dataset were scaled to have zero mean and unit variance. We also dropped the label column in the dataset. The architecture of the imputers is very simple. Here, we consider two variants: 1) Linear and 2) *Multi-Layer Perceptron* (MLP) with only two hidden layers. In the case of the target attribute is categorical, we use the Softmax activation in last layer. Table shows the specifications of imputers. The architecture of the discriminators is also very simple.

We implement them as MLP with only two hidden layers. The Relu activation is used in each hidden layer. Also, we used the sigmoid activation in the output layer. The architecture of the discriminators is shown in Table *1*.

Algorithm1. The proposed *UA-Adv Imputer*

```
Inputs: $\mathcal{D} = \{(\boldsymbol{x}_1, \boldsymbol{m}_1), (\boldsymbol{x}_2, \boldsymbol{m}_2), \dots, (\boldsymbol{x}_n, \boldsymbol{m}_n)\}$, $\lambda$ : controls the trade-off between the $\boldsymbol{\mathcal{L}}_{imp}$ and $\mathcal{L}_{fit}$ losses.
    1.Initialize the certainty matrix: $\boldsymbol{C} = \boldsymbol{M}$.
    2. Fill the $X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n]$ using the mean imputer.
    3. for iter = $1,2, \dots MAX\_Iter$
        3.1. for each $j \in S_{NA}$
            Freeze imputer $j$ and unfreeze discriminator $j$
            { Train discriminator j: $D(.;\boldsymbol{\phi}_j)$.}
            for $\ell = 1,2, \dots MAX\_Adv$
                Sample $\boldsymbol{X}^{real}$ from $\mathcal{D}$
                Generate $\boldsymbol{X}^{fake}$ by setting it equal to $\boldsymbol{X}^{real}$ and then adding noise
                    to the target values $\boldsymbol{X}^{fake}_{:j}$
                Compute $\mathcal{L}_{dis}(\boldsymbol{X}^{real}, \boldsymbol{X}^{fake})$ from (1).
                Backpropagate $\mathcal{L}_{dis}$ to optimize $\boldsymbol{\phi}_j$
            end;
            Update column $j$ of the certainty matrix $\boldsymbol{C}$: $\boldsymbol{C}_{:j} = [D(\boldsymbol{x}; \boldsymbol{\phi}_j)\ \textbf{for each}\ \boldsymbol{x}$ in $\boldsymbol{X}]$.
            { Train imputer j: $\mathrm{I}(.;\boldsymbol{\theta}_j)$.}
            Freeze discriminator j and unfreeze imputer j.
            for $\ell = 1,2, \dots MAX\_IMP$
                Sample $\boldsymbol{X}^{real}$ and $\boldsymbol{X}^{imp}$ from $\mathcal{D}$
                Fill $\boldsymbol{X}^{imp}_{:j}$ using $I(.;\boldsymbol{\theta}_j)$
                Compute $\mathcal{L}_{imp}(\boldsymbol{X}^{imp})$ from (3)
                Compute $\mathcal{L}_{fit}(\boldsymbol{X}^{real})$ from (6)
                $\mathcal{L}(\boldsymbol{X}^{real}, \boldsymbol{X}^{imp}) = \mathcal{L}_{fit}(\boldsymbol{X}^{real}) + \lambda \mathcal{L}_{imp}(\boldsymbol{X}^{imp})$,
                Backpropagate $\mathcal{L}$ to optimize $\boldsymbol{\theta}_j$
            end;
        end; { for each $j$ }
    end; { for iter = $1,2, \dots MAX\_Iter$ }
Output: Trained Imputers: $\mathcal{I} = \{I(;\boldsymbol{\theta}_j),\ j \in S_{NA}\}$.
```

Table 2. Specification and architecture of the imputers

Layer	Input	Output	Connected to	#Parameters
Linear Regressor				
FC	$d-1$	1	Inupt Data	d
Linear Classifier				
FC + Softmax	$d-1$	K	Inupt Data	dK
MLP Regressor				
FC1+Relu	$d-1$	$2(d-1)$	Input Data	$(d-1)(2d-1)$
FC2+Relu	$2(d-1)$	$d-1$	FC1	$(d-1)(2d-1)$
FC3	$d-1$	1	FC2	d
MLP Classifier				
FC1+Relu	$d-1$	$2(d-1)$	Input Data	$(d-1)(2d-1)$
FC2+Relu	$2(d-1)$	$d-1$	FC1	$(d-1)(2d-1)$
FC3+ Softmax	$d-1$	K	FC2	dK

FC: Fully Connected

Table 1. Specification and architecture of the discriminators

Layer	Input	Output	Connected to	#Parameters
MLP Classifier				
FC1+Relu	d	$2d$	Input Data	$2d(d+1)$
FC2+Relu	$2d$	d	FC1	$d(2d+1)$
FC3 + Sigmoid	d	1	FC2	$d+1$

5. Experimental results

This section deals with the experiments conducted to evaluate the effectiveness of the proposed imputer model.

5.1.1. Datasets

We evaluate the proposed model on four public UCI datasets [26]: 1) Breast Cancer Wisconsin Diagnostic (WDBC), 2) Parkinsons, 3) California, and 4) Yeast. The first two datasets are coming from the medical diagnosis domain which is considered as one of the main applications of imputing methods. The statistics of these datasets are summarized in Table 2.

Table 2. The Specifications of four real datasets used in our experiments.

Data Set	#Classes	n	d	Description
WDBC	2	569	30	Breast Cancer Wisconsin (Diagnostic) dataset from the University of California- Irvine. The aim of the data is to discriminate healthy people from those with cancer disease. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)
Parkinsons	2	195	23	The dataset is collected of a range of biomedical voice measurements from 31 individuals where 23 have Parkinson's disease. Each feature shows a particular voice measure, and each row corresponds to one person. The target of the data is to discriminate healthy people from those with Parkinson's disease. https://archive.ics.uci.edu/ml/datasets/parkinsons
California	Real between [0.15 – 5]	20640	8	Data are drawn from the 1990 U.S. Census. The target variable shows the median house value in the logarithmic scale. We obtained it from the *sklearn.datasets* package.
Yeast	10	1484	8	The Yeast dataset include protein-protein interactions. The aim is to predict localization site of protein. https://archive.ics.uci.edu/ml/datasets/Yeast

5.1.2. Evaluation metrics

There are some standard metrics to evaluate the performance of an imputation system. *Root Mean Square Error* (RMSE) and *Mean Absolute Error* (MAE) are two measures that show the difference between imputed and real-values. These metrics used in most studies, are also adopted in our experiments.

$$RMSE = \frac{1}{n}\sqrt{\sum_{i=1}^{n}\left(X_i^{true} - X_i^{imp}\right)^2} \tag{9}$$

$$MAE = \frac{1}{n}\left|X_i^{true} - X_i^{imp}\right| \tag{10}$$

6. Experimental setup

We compared the proposed *UA-Adv Imputer* with four typical and state-of-the-art algorithms named:

1. **OT-Imputer** [15]: two variants are considered in the experiments: 1) Linear and 2) MLP. These variants are built as described in the original paper.
2. **Soft Impute** [4]: that is a low-rank approximation method that imputes missing entries by performing iterative *Singular Value Decomposition* (SVD).
3. Mean Imputer.
4. **ICE** (Imputation by Chained equations) [27]: that performs imputation using conditional expectation. This method is implemented in the *scikit-learn* python package. ICE is considered one the most popular method due to providing good imputations with little hyperparameter adjustment.

We refer to these algorithms as peer methods. Additionally, we compare our methods to three deep learning-based methods:

1. **MIWAE** [28]: that optimizes *Importance Weighted AutoEncoder* (IWAE) to impute missing information.
2. **GAIN** [13]: that adapts GAN models to the data imputation task.
3. **VAEAC** [29]: that extends *Variational Auto Encoder* (VAE) so that they can be conditioned on an arbitrary observed data. The missing data are then filled by sampling from the trained VAEs.

We split each dataset 80/20 (train/test) and adopt k-fold (k=5) cross-validation to adjust the hyperparameters of the methods. More specifically, we chose the learning rate (lr) from the range:$\{10^{-4}, 5\times10^{-4}, 10^{-3}\}$, optimizer from {Adam, RmsProp}. The hyperparameters of the competing methods along with their adjustment are reported in

Table 3.

To generate missing patterns, we selected $p_{fea} = 30\%$ of columns as missing. For each missing column, we randomly chose $p = 50\%$ of values and set them equal to $None$ (missing).

Table 3. Hyperparameters of the competing imputers along their adjustments

Hyper-parameter	**Method(s)**	**Description**	**Range**
MAX_Iter	OT-Imputer, UA-Adv	Number of cycles that each feature optimized	$\{10,15,20\}$
λ	UA-Adv	Balance factor between the two loss terms	$\{0.1, 0.5,1,2,3,5\}$
λ	Soft Impute	Nuclear-norm regularization parameter	A grid containing 15 values between λ_{min} and λ_{max}

Table 4. MAE of Imputation methods on the datasets.

Methods\Datasets	WDBC	California	Parkinson	Yeast
Linear UA-Adv	**0.193**	0.251	0.341	0.720
MLP UA-Adv	0.209	**0.213**	**0.302**	**0.712**
OT Imputer Linear	0.298	0.292	0.375	0.723
MLP OT Imputer	0.322	0.251	0.344	0.718
Mean	0.725	0.683	0.796	0.740
ICE	0.210	0.254	0.392	0.751
Soft Impute	0.240	0.351	0.370	0.726

Table 5. RMSE of Imputation methods on the datasets

Methods\Datasets	WDBC	California	Parkinson	Yeast
Linear UA-Adv	**0.358**	0.372	0.653	1.011
MLP UA-Adv	0.369	**0.336**	**0.636**	**0.999**
Linear OT Imputer	0.478	0.621	0.651	1.018
MLP OT Imputer	0.495	0.550	0.640	1.017
Mean	1.002	0.992	1.034	1.036
ICE	0.381	0.363	0.711	1.046
Soft Impute	0.401	0.546	**0.636**	1.015

6.1. Comparison with peer methods

The results of proposed models on test data along with comparisons with peer methods are reported in

Table *4* and Table 5. Also, we plot the MAE and RMSE of the competing methods versus iterations in Figure 7 and Figure 8 on the evaluated datasets respectively.

7. Discussion

As the results indicate, the proposed models outperform the peer methods in most of the evaluated datasets by a large margin. It confirms the efficacy of the proposed loss model, novel training strategy of the discriminators, and considering uncertainty in the training process of imputers. Through the ablation study provided in the next experiments, we carefully examine the contribution of each factor in the overall performance of the proposed models. Besides, as expected, the mean imputer achieved the worst results due to neglecting the dependency between features. Additionally, the state-of-the-art OT Imputer did not obtain competitive results. It can be explained by the fact that this method only focused on minimizing the distribution difference between two imputed samples and neglects the importance of the accuracy of imputers to fill the missing entries. In contrast, the proposed models train accurate imputers via the $\mathcal{L}_{fit}$loss term.

Finally, in both UA-Adv and OT imputers, the MLP models almost outperform the linear ones due to their ability to capture the non-linear relationship between features needed for the imputation task.

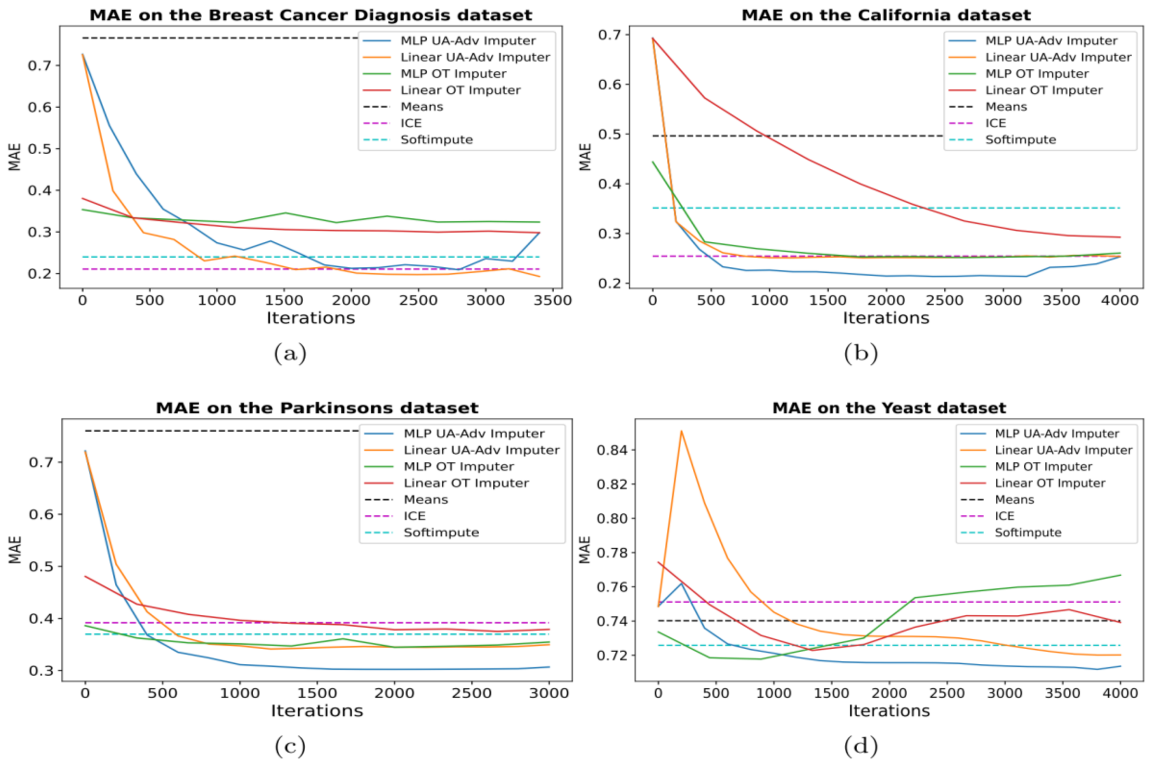


Figure 7.MAE of the imputers on the datasets

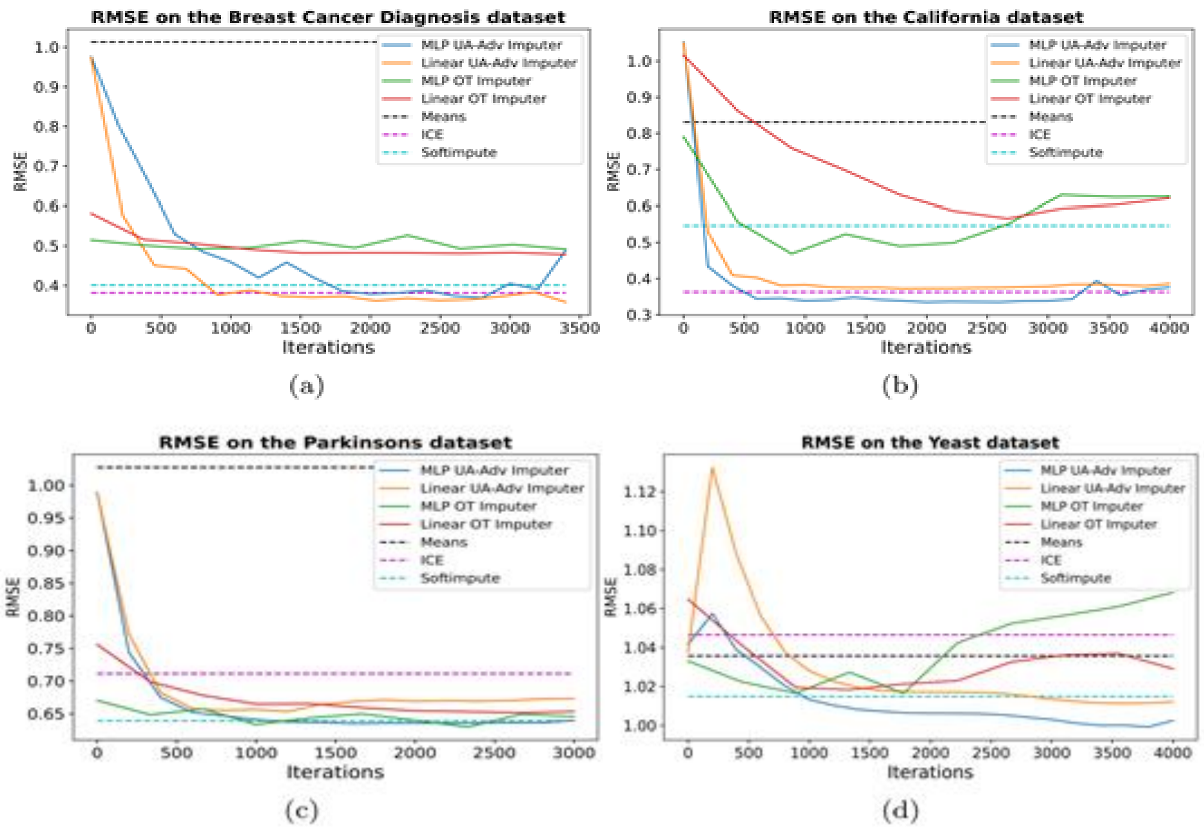
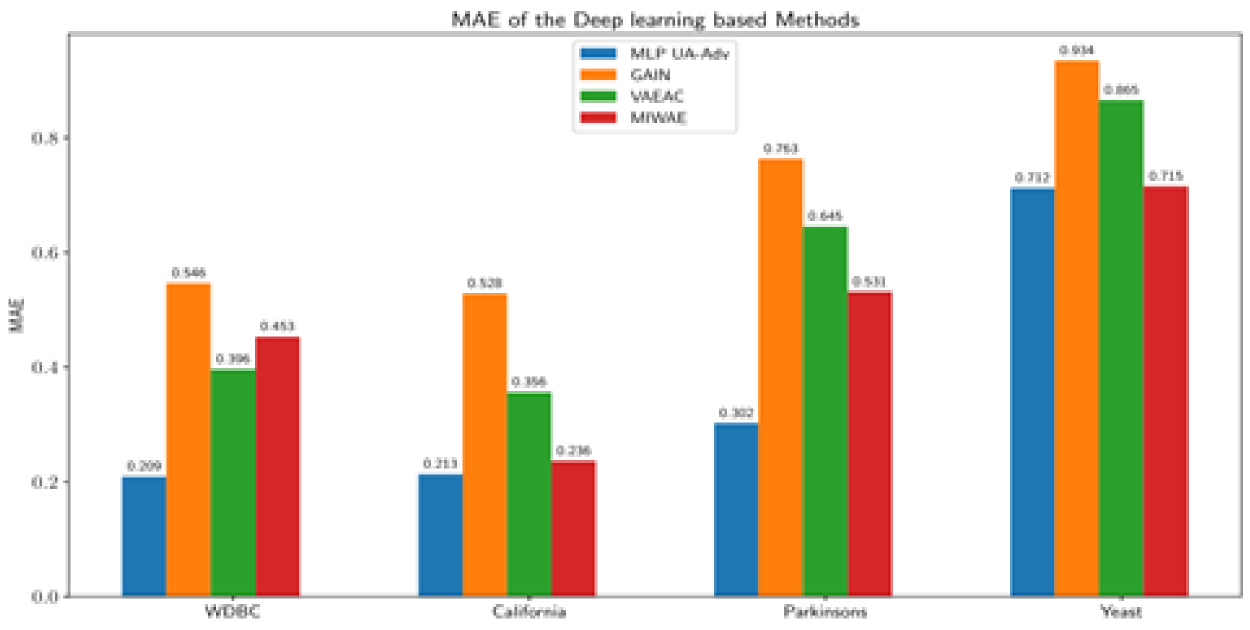


Figure 8. RMSE of the imputers on the datasets

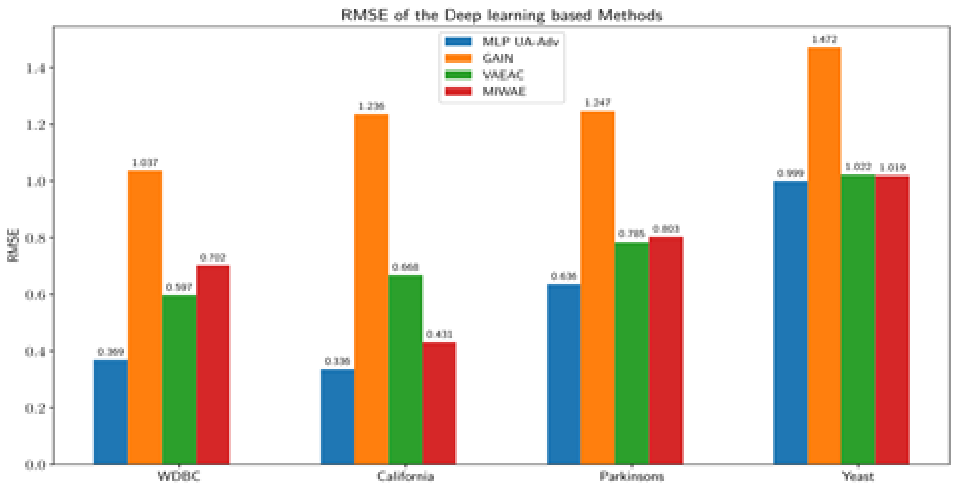
7.1. Comparison with deep learning-based methods

Figure 9 depicts the results of proposed methods and the competing deep learning-based imputer. Comparison with deep learning methods indicates that the proposed methods consistently outperform them in all evaluated datasets. Indeed, deep learning methods results are not even as good as baseline models such as ICE and Soft

Impute. It can be explained as these methods are originally developed for large datasets and custom data types such as images, video, and text. Thus, adopting these methods for small and tabular datasets used in our experiments is not straightforward and as seen yielded poor results.



(a)



(b)

Figure 9. MAE and RMSE of the deep learning-based imputers in comparison with the proposed MLP UA-Adv Imputer

7.1.1. Effects of the adversarial module

The hyperparameter λ in the proposed loss function controls the influence of the adversarial module in the training process of imputers. Thus, to investigate the effectiveness of the module, we change λ from small to large values and plot the MAE and RMSE of the *MLP UA-Adv Imputer* vs λ on the Parkinsons and Yeast datasets in Figure 10. Besides, the results were compared to the Soft Impute which provides a better insight into the sensitivity of results to this hyperparameter. As the result indicate, the performance of *MLP UA-Adv Imputer* maximized in the range [1,2] and [0.5,1] on the Parkinsons and Yeast datasets respectively. Also, a small value of λ leads to unsatisfactory results in both datasets that reveals the importance of the adversarial module. Additionally, setting λ above the optimal value decreases the performance of the imputer slightly but still, the proposed imputer outperforms the Soft-impute by a large margin. We can conclude that the adversarial module plays an important role in the overall performance of the proposed model. Also, the results over the large subset of λ values are acceptable and vary smoothly that making the adjustment of this hyperparameter a straightforward task.

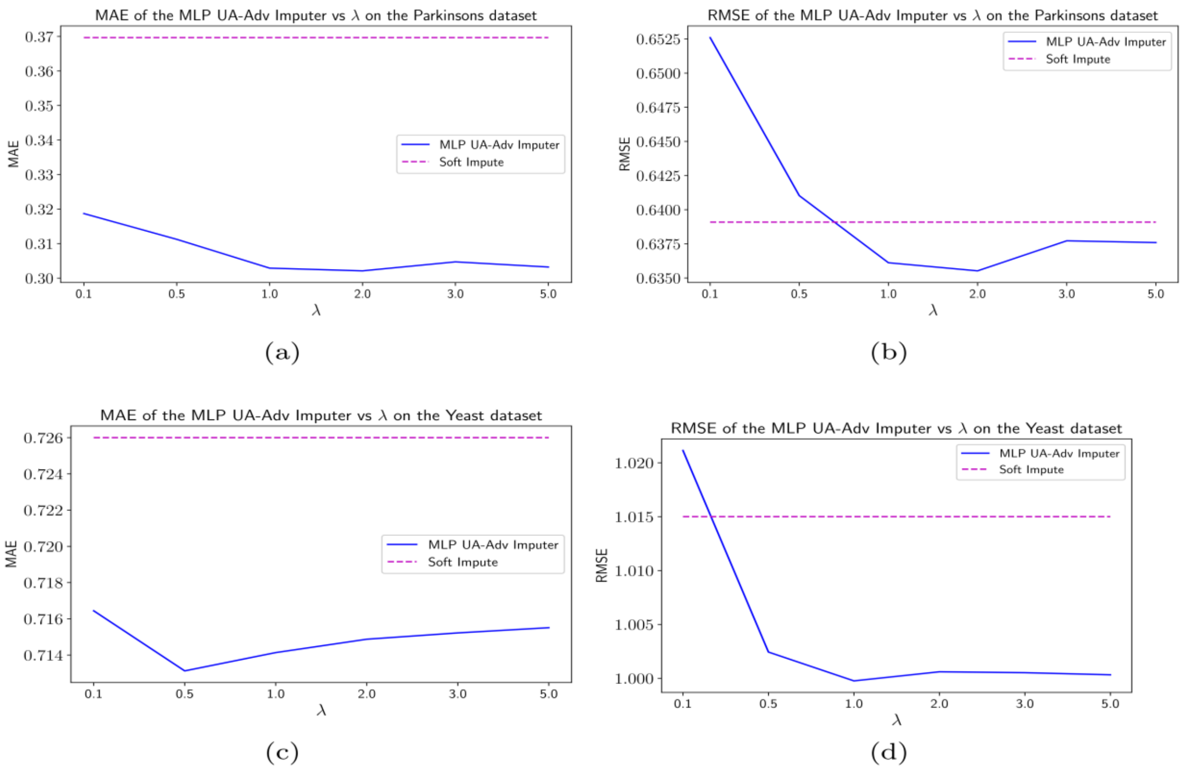


Figure 10. Influence of the adversarial module in the performance of the MLP UA-Adv

7.1.2. Effects of the uncertainty scores

To examine the role of uncertainty scores for imputed values in the overall performance of the proposed model, we consider a variant of *MLP UA-Adv* Imputer named *MLP Adv* Imputer that does not consider the uncertainty scores provided by the adversarial module. The results obtained by *MLP Adv* Imputer are compared with *MLP UA-Adv* Imputer in Figure 11. As the results indicate, in both evaluated datasets *MLP UA-Adv Imputer* outperforms *MLP Adv Imputer* considerably. It reveals the importance of considering uncertainty scores in the training of the imputers. Also, we observed that the optimal value of hyperparameter λ in *MLP Adv* is much greater than *MLP UA-Adv* in both datasets. That indicates weighting the features by uncertainty scores yields more reliable input data for training the imputers using regression or classification loss (i.e., $\mathcal{L}_{fit}$ loss term).

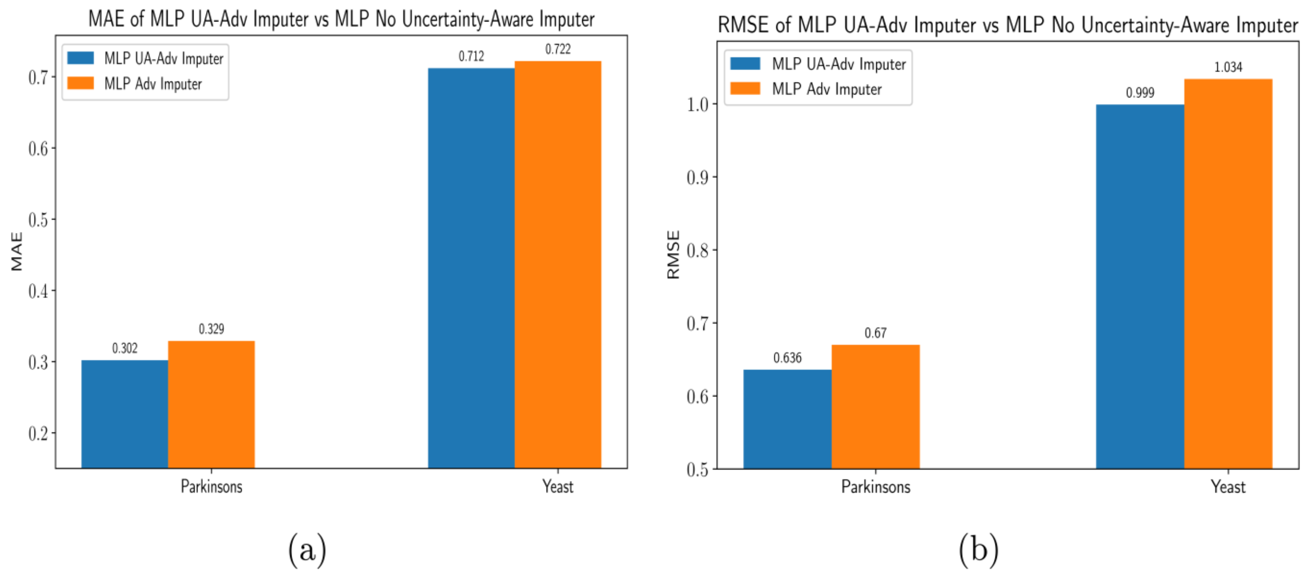


Figure 11. Effects of Uncertainty Scores in the performance of MLP UA-Adv Imputer

7.1.3. Run-time analysis

We compared the runtime of both proposed *MLP UA-Adv* and *Linear UA-Adv* methods with that of *MLP OT* and *Linear OT* imputers on some datasets. All methods are implemented with the *PyTorch* deep learning library and have been executed on a computer with the GTX 1660-ti (6 GB) graphic card and the Intel 9750H CPU. The execution time needed to optimize the imputer is plotted in Figure 12. As the results show, the proposed method has considerably less runtime compared to OT imputers. The main reason is that OT methods require computing the *Sinkhorn* divergence between two random batches in the training process which is inefficient in terms of computational cost. Also, the time required to train the MLP imputers is almost the same as linear ones and the extra overhead is negligible. Thus, considering the MLP imputers in the proposed model consistently outperform the linear counterpart, they are the preferred models for the imputation task in practice.

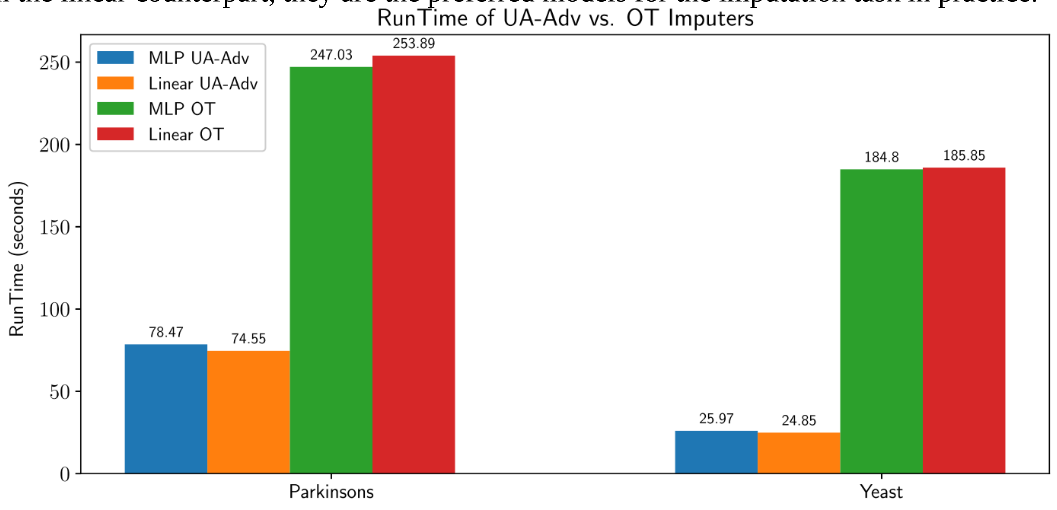


Figure 12. Runtime of the proposed methods vs. OT imputers

8. Conclusion and future work

In this research, we study the imputation for missing datasets. The aims were to train uncertainty-aware imputers and boost their performance using modern regularization techniques in the deep learning domain. For these reasons, we develop a simple neural network-based architecture that can train well with small datasets and utilizes a novel adversarial strategy to estimate the uncertainty of imputed data. Besides, we proposed a novel hybrid loss function that enforces the imputers to generate values for missing data that on the one hand, obey the underlying data distribution so that it can confuse the well-trained adversarial module, and on the other hand, predict existing non-missing values accurately. Experiments conducted on four real datasets collected from the UCI repository reveal that the proposed imputers are indeed effective and surpass the peer methods by a large margin almost on all evaluated datasets. Besides, we carefully examined the contribution of the adversarial module and the uncertainty scores through ablation studies. The results confirm that both considerably boost the overall performance of our methods. Finally, the run time of the proposed methods was investigated, and the results show that they are efficient and have less execution time in comparison with that of peer imputer models. In future work, we aim to extend our work to take the imbalanced nature of the imputation task into account. Also, we target to examine other methods for incorporating the uncertainty scores in the training process of imputers.

Declaration of competing interest

The authors declare that they have no any known financial or non-financial competing interests in any material discussed in this paper.

Funding information

No funding was received from any financial organization to conduct this research.

References

[1] R. J. Little and D. B. Rubin, *Statistical analysis with missing data*. John Wiley & Sons, 2019.

[2] Z. Zhu, T. Wang, and R. J. Samworth, "High-dimensional principal component analysis with heterogeneous missingness," *arXiv preprint arXiv:1906.12125,* 2019.

[3] S. Van Buuren, *Flexible imputation of missing data*. CRC press, 2018.

[4] T. Hastie, R. Mazumder, J. D. Lee, and R. Zadeh, "Matrix completion and low-rank SVD via fast alternating least squares," *The Journal of Machine Learning Research,* vol. 16, no. 1, pp. 3367-3402, 2015.

[5] L. P. Brás and J. C. Menezes, "Improving cluster-based missing value estimation of DNA microarray data," *Biomolecular engineering,* vol. 24, no. 2, pp. 273-282, 2007.

[6] K.-Y. Kim, B.-J. Kim, and G.-S. Yi, "Reuse of imputed data in microarray analysis increases imputation efficiency," *BMC bioinformatics,* vol. 5, no. 1, pp. 1-9, 2004.

[7] A. S. Khairy, H. Salim, "The Detection of Counterfeit Banknotes Using Ensemble Learning Techniques of AdaBoost and Voting," *International Journal of Intelligent Engineering and Systems,* vol. 14, no. 1, pp. 326-339, 2021.

[8] M. J. Azur, E. A. Stuart, C. Frangakis, and P. J. Leaf, "Multiple imputation by chained equations: what is it and how does it work?," *International journal of methods in psychiatric research,* vol. 20, no. 1, pp. 40-49, 2011.

[9] Z. Ghahramani and M. I. Jordan, "Supervised learning from incomplete data via an EM approach," in *Advances in neural information processing systems*, 1994, pp. 120-127.

[10] D. Bertsimas, C. Pawlowski, and Y. D. Zhuo, "From predictive methods to missing data imputation: an optimization approach," *The Journal of Machine Learning Research,* vol. 18, no. 1, pp. 7133-7171, 2017.

[11] L. Gondara and K. Wang, "Mida: Multiple imputation using denoising autoencoders," in *Pacific-Asia conference on knowledge discovery and data mining,* 2018, pp. 260-272: Springer.

[12] S. C.-X. Li, B. Jiang, and B. Marlin, "Misgan: Learning from incomplete data with generative adversarial networks," *arXiv preprint arXiv:1902.09599,* 2019.

[13] J. Yoon, J. Jordon, and M. Schaar, "Gain: Missing data imputation using generative adversarial nets," in *International Conference on Machine Learning*, 2018, pp. 5689-5698: PMLR.

[14] M. Ahmadi, T. Nest, M. Abdelnaim, and T.-D. Le, "Reproducing AmbientGAN: Generative models from lossy measurements," *arXiv preprint arXiv:1810.10108,* 2018.

[15] B. Muzellec, J. Josse, C. Boyer, and M. Cuturi, "Missing data imputation using optimal transport," in *International Conference on Machine Learning*, 2020, pp. 7130-7140: PMLR.

[16] A. W. Mulyadi, E. Jun, and H.-I. Suk, "Uncertainty-aware variational-recurrent imputation network for clinical time series," *IEEE Transactions on Cybernetics,* 2021.

[17] H. T. S. A. Abdul Hadi M.Alaidi Ibtisam A. Aljazaery, "Encryption of Color Image Based on DNA Strand and Exponential Factor," *International Journal of Online and Biomedical Engineering (iJOE),* vol. 18, no. 3, 2022.

[18] T. H. Bø, B. Dysvik, and I. Jonassen, "LSimpute: accurate estimation of missing values in microarray data with least squares methods," *Nucleic acids research,* vol. 32, no. 3, pp. e34-e34, 2004.

[19] X. Wang, . Li, Z. Jiang, and H. Feng, "Missing value estimation for DNA microarray gene expression data by Support Vector Regression imputation and orthogonal coding scheme," *BMC bioinformatics,* vol. 7, no. 1, pp. 1-10, 2006.

[20] L. F. Burgette and J. P. Reiter, "Multiple imputation for missing data via sequential regression trees," *American journal of epidemiology,* vol. 172, no. 9, pp. 1070-1076, 2010.

[21] D. J. Stekhoven and P. Bühlmann, "MissForest—non-parametric missing value imputation for mixed-type data," *Bioinformatics,* vol. 28, no. 1, pp. 112-118, 2012.

[22] R. Mazumder, T. Hastie, and R. Tibshirani, "Spectral regularization algorithms for learning large incomplete matrices," *The Journal of Machine Learning Research,* vol. 11, pp. 2287-2322, 2010.

[23] S. Mohamed, Z. Ghahramani, and K. A. Heller, "Bayesian exponential family PCA," *Advances in neural information processing systems,* vol. 21, pp. 1089-1096, 2008.

[24] R. Lall, "How multiple imputation makes a difference," *Political Analysis,* vol. 24, no. 4, pp. 414-433, 2016.

[25] J. Honaker, G. King, and M. Blackwell, "Amelia II: A program for missing data," *Journal of statistical software,* vol. 45, no. 1, pp. 1-47, 2011.

[26] M. Lichman. (2013). *UCI Machine Learning Repository University of California, Irvine, School of Information and Computer Sciences*. Available: http://archive.ics.uci.edu/ml

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, and V. Dubourg, "Scikit-learn: Machine learning in Python," *the Journal of machine Learning research*, vol. 12, pp. 2825-2830, 2011.

[28] P.-A. Mattei and J. Frellsen, "MIWAE: Deep generative modelling and imputation of incomplete data sets," in *International conference on machine learning*, 2019, pp. 4413-4423: PMLR.

[29] O. Ivanov, M. Figurnov, and D. Vetrov, "Variational autoencoder with arbitrary conditioning," *arXiv preprint arXiv:1806.02382*, 2018.