SGAIN, WSGAIN-CP and WSGAIN-GP: Novel GAN Methods for Missing Data Imputation

Diogo Telmo Neves[1,2,3], Marcel Ganesh Naik[2] ⋆., and Alberto Proença[3]

[1] Intelligent Analytics for Mass Data (IAM)
German Research Center for Artificial Intelligence (DFKI), Berlin, Germany
[2] Charité – Universitätsmedizin, and Berlin Institute of Health, Berlin, Germany
{diogo-telmo.neves, marcel.naik}@charite.de
[3] Centro ALGORITMI, Universidade do Minho, Braga, Portugal
{dneves, aproenca}@di.uminho.pt

Abstract. Real-world datasets often have missing values, which hinders the use of a large number of machine learning (ML) estimators. To overcome this limitation in a data analysis pipeline, data points may be deleted in a data preprocessing stage. However, an alternative better solution is data imputation.
Several methods based on Artificial Neural Networks (ANN) have been recently proposed as successful alternatives to classical discriminative imputation methods. Amongst those ANN imputation methods are the ones that rely on Generative Adversarial Networks (GAN).
This paper presents three data imputation methods based on GAN: SGAIN, WSGAIN-CP and WSGAIN-GP. These methods were tested on datasets with different settings of missing values probabilities, where the values are missing completely at random (MCAR). The evaluation of the newly developed methods shows that they are equivalent or outperform competitive state-of-the-art imputation methods in different ways, either in terms of response time, the data imputation quality, or the accuracy of post-imputation tasks (e.g., prediction or classification).

Keywords: Missing Data · Data Imputation · Generative Adversarial Network.

1 Introduction

Real-world datasets often have missing values, a large number of those can be found at the website of OpenML [23] or at the Machine Learning (ML) repository maintained by the University of California at Irvine [6]. TBase [14], one of the largest transplant databases in Europe, is another example of a large collection of data with missing values. A few reasons for the missing values in TBase are: a sensor malfunction, a nurse that forgot to register the weight of a patient, or an attribute (of the database) that accepts Null entries. Yet, domain experts

⋆ The second author is a participant in the BIH Charité Digital Clinician Scientist Program funded by the Charité – Universitätsmedizin Berlin, the Berlin Institute of Health and the German Research Foundation (DFG)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_10

often find incorrect values for a given attribute (aka variable or feature), due to quantitative or qualitative data errors [2, 1]. Incorrect values may also be marked as missing values, turning a complete dataset into an incomplete one or an already incomplete dataset may end up with more missing values.

Datasets with missing values will hinder the use of a large number of machine learning (ML) estimators and/or will impair the ML model quality. As a consequence, the conclusions and insights that could be extracted from the data may be of no use [17, 9, 12, 22, 15]. Handling missing values is a common task during the data preprocessing stage of a data analysis pipeline [2, 1], which has challenged many researchers. One possibility is to apply *listwise deletion* (i.e., remove the data points that have at least one missing value) as a way to get a complete dataset from an incomplete one. However, listwise deletion is not always an adequate solution, since it may lead to a high decrease in the amount of data points and, ultimately, to an empty dataset, which would turn impracticable or meaningless the planned data analysis [17, 9, 12, 22, 15].

A substantial effort has been dedicated to devise new algorithms, methods, libraries and frameworks for robust data imputation, from univariate to multivariate techniques, from basic imputation (e.g., mean, median and mode) to regression-based algorithms (e.g., linear, logistic, or stochastic regression), from discriminative to generative imputation methods. Among the latter, the Generative Adversarial Networks (GAN) became popular for its extraordinary capability of capturing the data distribution. GAN is the base for our novel data imputation methods, and also used in *purify* for synthetic data generation[4].

The mechanisms of missingness are typically classified as missing at random (MAR), missing completely at random (MCAR) and missing not at random (MNAR). A precise definition of these terms can be found in [18]. This work only addresses the MCAR mechanism, mainly due to: (i) modelling MAR or MNAR data requires domain knowledge as well as a deep insight on every detail of the data, hardly feasible with third-party datasets, while in MCAR the missingness does not depend on the observed data, nor on the unobserved data; (ii) modelling MCAR data requires less complex ML models, since no feature (attribute or variable) dependency needs to be modeled; (iii) literature describing the main imputation methods (discriminative or generative) usually assume that missing data is MCAR; (iv) by assuming the same missingness setting, fair comparisons of our work against competitive methods can be established.

Current references in generative imputation methods are GAIN [24], and a Wasserstein GAIN (WGAIN) [7]. GAIN achieved excellent results due to the exceptional ability of its GAN-based architecture to generate a model data distribution close to the real data distribution [8, 24]; however, it has some caveats, namely the optimization process is delicate and unstable, as theoretically shown in [3], and the training phase is a computationally expensive task [20, 11].

[4] The first author used synthetic data to develop and test ML models for the kidney disease pilot (see https://www.bigmedilytics.eu/pilot/kidney-disease/) of BigMedilytics (an EU-funded project supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 780495).

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_10

The key contributions of this work are the three novel generative imputation methods, all improved versions of the GAIN implementation: a *Slim* GAIN (SGAIN), a *Wasserstein Slim* GAIN *with Clipping Penalty* (WSGAIN-CP) and a *Wasserstein Slim* GAIN *with Gradient Penalty* (WSGAIN-GP).

Other contributions include:

- concrete and scientific evidence (in Section 2) that the algorithm described in Section 3.3, is a trustworthy implementation of a Wasserstein GAN and has a better overall quality when compared to the one described in [7];
- an empirical experimental demonstration (in Section 4) that the novel methods outperform state-of-the-art imputation methods in different aspects, namely in response time, on data imputation quality, or in the accuracy of post-imputation tasks (e.g., prediction or classification).

2 Imputation Methods Based on GAN

Imputation methods can be classified as discriminative or generative methods. Classical imputation methods typically fall in the former class, while advanced methods based on ANN fall in the latter. Classical methods, such as KNNImpute [21], MICE [5] and MissForest [19] are mature and have a wide acceptance, being used in several domains or applications. Typically, the use of these methods is grounded on assumptions about the data itself, namely on data distributions, correlations, skewness, and dependencies. On the other hand, generative methods rely on a different approach, no matter if for data imputation or not: to learn from data samples.

Recently, Goodfellow *et al.* devised a novel deep learning architecture, GAN, based on two ANN that contest each other, a *generator* and a *discriminator* [8], which became known as *a minimax two-player game*. Despite their impressive capability to learn from data samples and to capture the data distribution, they have undesired features, namely mode collapse and vanishing gradients.

To mitigate these GAN caveats, a Wasserstein GAN was proposed [4], which impose bounds on the weights of the discriminator ANN (renamed as *critic*). Few months later an improvement to the training of a Wasserstein GAN was also proposed [10]. A little after, Soon Yoon *et al.* proposed GAIN [24], a simple but yet very effective and accurate data imputation method based on a GAN. However, since GAIN is based on a vanilla GAN, it still has the aforementioned caveats (besides the ones mentioned in Section 1).

To reduce the impact of those limitations, we decided to develop a *slim* Wasserstein GAIN and explore variants with two types of penalties: a *weight clipping penalty* (WSGAIN-CP) and a *gradient penalty* (WSGAIN-GP). These data imputation methods exhibit better execution times than the GAIN counterparts.

Very recently, Friedjungová *et al.* published a work [7] that claims to have implemented a data imputation method using a Wasserstein GAN [7]. Unfortunately, there is no public repository with that implementation, which hinders a thoroughly comparison against such work. Furthermore, in our opinion, the algorithm described in [7] cannot be considered a WGAN [4] since, for instance, the

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_10

critic ANN is not trained more times than the generator ANN. Additionally, the results presented in [7] do not promote a fair comparison with those published in [24] and several inconsistencies and discrepancies are noticed. Therefore, we consider that our WSGAIN-CP data imputation method is a trustworthy implementation of a Wasserstein GAN and has a better overall quality when compared to the one described in [7].

3 Novel Generative Imputation Methods

This section introduces our novel generative imputation methods: SGAIN, derived from the implementation of GAIN [24] and grounded on the seminal work of Goodfellow *et al.* [8]; WSGAIN-CP and WSGAIN-GP, Wasserstein variations of the baseline data imputation method, SGAIN. Before introducing these three generative imputation methods, we start paving the way with some notation and giving a brief problem formulation to ease the understanding of each algorithm.

3.1 Notation and Problem Formulation

In terms of linear algebra operations, let us consider $\odot$, $\oplus$, and $\ominus$ as the element-wise multiplication, addition, and subtraction operations, respectively.

Let us consider:

- $\mathcal{X}$ an $\mathbb{R}^d$ data space: $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_d$;
- d independent continuous and/or discrete random variables $\mathbf{X} = (X_1, \ldots, X_d)$ that take values in $\mathcal{X}$ and whose distribution is denoted by $P(\mathbf{X})$;
- $\mathbf{M} = (M_1, \ldots, M_d)$ as being a mask of the values in $\mathbf{X}$, where *a missing value* in $\{X_i\}_{i=1}^d$ is represented by a zero and any non-missing value is represented by a one; thus, $\{M_i\}_{i=1}^d$ assume values in $\{0, 1\}$.

The goal of the imputation process is to estimate values for all missing values in each $\{X_i\}_{i=1}^d$. However, it could happen that the estimation is done for every element, no matter if it corresponds to a missing value or not. Generative imputation methods (e.g., [24]) do exactly this, since they attempt to model the distribution of the (existing) data (i.e., $P(\mathbf{X})$).

Let us denote $\bar{\mathbf{X}}$ as the output vector of an abstract imputation function that produces estimations for all elements in each $\{X_i\}_{i=1}^d$. Then, the estimation final vector, which we denote as $\hat{\mathbf{X}}$, can be assembled as follows:

$$\hat{\mathbf{X}} = \mathbf{M} \odot \mathbf{X} \oplus (1 \ominus \mathbf{M}) \odot \bar{\mathbf{X}} \tag{1}$$

3.2 Slim GAIN

This section introduces the *Slim GAIN* imputation method (SGAIN). The architecture of SGAIN (in Figure 1) is derived from the counterpart of GAIN [24].

In contrast to the architecture of GAIN, in SGAIN there is no Hint Generator and, consequently, no Hint Matrix is generated. The architecture of SGAIN is even slimmer, since both the generator and the discriminator neural networks have only two layers, whereas in GAIN each of them has three layers.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_10

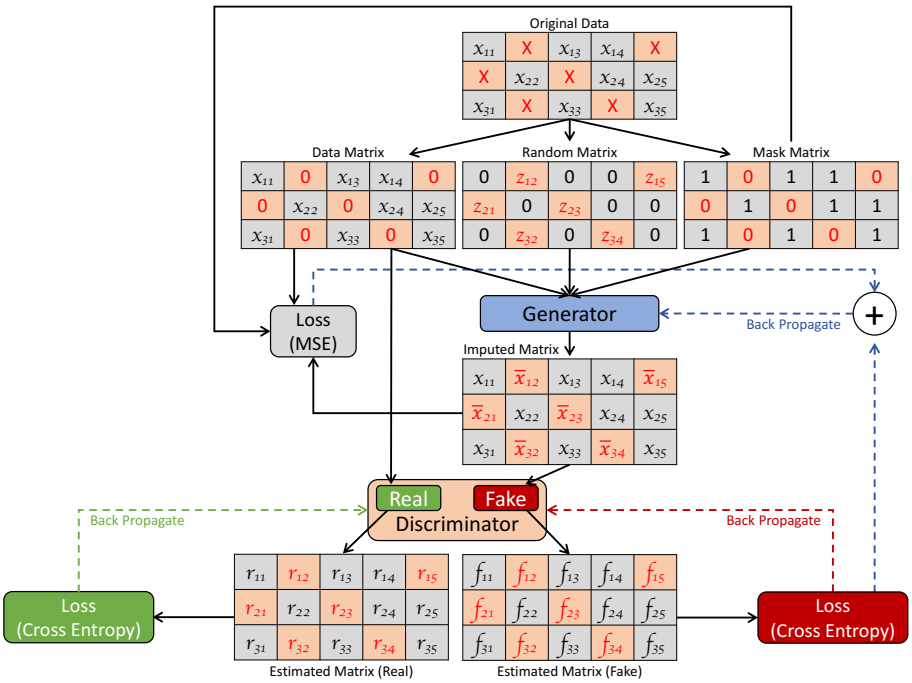

Fig. 1: The architecture of SGAIN.

SGAIN uses the hyperbolic tangent activation function (aka `tanh`) in the output layers of the generator and the discriminator. The rationale to use the `tanh` and not the `sigmoid` activation function is twofold: (i) the convergence of the optimizer used in a neural network is usually faster if its inputs are linearly transformed to have zero means and unit variances, and decorrelated, as discussed in [13]; and (ii) the derivatives of the `tanh` activation function are larger than the derivatives of the `sigmoid`, which means that the optimizer can converge faster when `tanh` is used.

The architecture of SGAIN also invokes twice the discriminator, one for the *real* data and the other for the *fake* data. This brings the architecture of SGAIN closer to the one used by Goodfellow *et al.* in [8].

Generator The generator (G) inputs are $\mathbf{Z}$ and $\mathbf{M}$, and the output is $\bar{\mathbf{X}}$. $\mathbf{Z}$ is a d-dimensional variable, $\mathbf{Z} = (Z_1, \ldots, Z_d)$, in which each $\{Z_i\}_{i=1}^d$ has the non-missing values of $\{X_i\}_{i=1}^d$ and the missing values in $\{X_i\}_{i=1}^d$ are replaced by random values (aka noise). We denote $\mathbf{N} = (N_1, \ldots, N_d)$ as the output of a function that draws out random values from a continuous uniform distribution (a common configuration is to use the interval $[-0.01, +0.01]$).

Formally, the $\mathbf{Z}$ vector is assembled as follows:

$$\mathbf{Z} = \mathbf{M} \odot \mathbf{X} \oplus (1 \ominus \mathbf{M}) \odot \mathbf{N} \tag{2}$$

$\bar{\mathbf{X}}$ is formally defined as:

$$\bar{\mathbf{X}} = G(\mathbf{Z}, \mathbf{M}) \tag{3}$$

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_10

Discriminator The discriminator (D) is the other player of the *minimax* game described in [8]. In SGAIN, the discriminator input is either the *real* data (i.e., $\mathbf{X}$) or the *fake* data (i.e., $\bar{\mathbf{X}}$); the fake data is produced by G (Equation 3).

Using back-propagation, the different outputs of D are used to compute the losses of the generator and of the discriminator.

Algorithm 1: Pseudo-code of SGAIN

Input: $\mathbf{X}$ // dataset w/ missing values
1 **Parameter:** mb // mini-batch size
2 **Parameter:** α // hyper-parameter of Generator loss
3 **Parameter:** n_iter // number of iterations
4 **Result:** $\hat{\mathbf{X}}$ // imputed dataset
5 $\mathbf{M} \leftarrow mask(\mathbf{X})$ // each miss. value is 0, otherwise 1
6 **for** *iter* $\leftarrow 1$ *to* *n_iter* **do**
7 Draw mb samples from $\mathbf{X}$: $\{\tilde{\mathbf{x}}(j)\}_{j=1}^{mb}$
8 Draw mb samples from $\mathbf{M}$: $\{\tilde{\mathbf{m}}(j)\}_{j=1}^{mb}$
9 Draw mb i.i.d. samples of $\mathbf{N}$: $\{\tilde{\mathbf{n}}(j)\}_{j=1}^{mb}$ // random noise
10 **for** $j \leftarrow 1$ *to* mb **do**
11 $\tilde{\mathbf{z}}(j) \leftarrow \tilde{\mathbf{m}}(j) \odot \tilde{\mathbf{x}}(j) \oplus \left(1 \ominus \tilde{\mathbf{m}}(j)\right) \odot \mathbf{n}(j)$ // Equation 2
12 $\bar{\mathbf{x}}(j) \leftarrow G\left(\tilde{\mathbf{z}}(j), \tilde{\mathbf{m}}(j)\right)$ // Equation 3
13 **end**
// (1) Discriminator Optimization
14 Update D using Adam or RMSprop or SGD
15 $\nabla_{\theta_D} - \frac{1}{mb}\sum_{j=1}^{mb}\mathcal{L}_D\left(D\left(\tilde{\mathbf{x}}(j)\right), D\left(\bar{\mathbf{x}}(j)\right), \tilde{\mathbf{m}}(j)\right)$
// (2) Generator Optimization
16 Update G using Adam or RMSprop or SGD
17 $\nabla_{\theta_G} - \frac{1}{mb}\sum_{j=1}^{mb}\mathcal{L}_G\left(D\left(\bar{\mathbf{x}}(j), \tilde{\mathbf{m}}(j)\right)\right) + \frac{\alpha}{mb}\sum_{j=1}^{mb}\mathcal{L}_{MSE}\left(\tilde{\mathbf{x}}(j), \bar{\mathbf{x}}(j), \tilde{\mathbf{m}}(j)\right)$
18 **end**
19 $\mathbf{Z} \leftarrow \mathbf{M} \odot \mathbf{X} \oplus (1 \ominus \mathbf{M}) \odot \mathbf{N}$ // Equation 2
20 $\bar{\mathbf{X}} \leftarrow G(\mathbf{Z}, \mathbf{M})$ // Equation 3
21 $\hat{\mathbf{X}} \leftarrow \mathbf{M} \odot \mathbf{X} \oplus (1 \ominus \mathbf{M}) \odot \bar{\mathbf{X}}$ // Equation 1

SGAIN Algorithm A common step during the training of a GAN is to draw samples from the (training) data, which form a mini-batch of data used in an iteration. The SGAIN algorithm, as shown in Algorithm 1, also has this step. To fully understand the algorithm let us denote $\tilde{\mathbf{x}}$, $\tilde{\mathbf{m}}$, and $\tilde{\mathbf{n}}$ as samples draw from $\mathbf{X}$, $\mathbf{M}$, and $\mathbf{N}$, respectively, and for the same data points.

In a nutshell, the SGAIN algorithm follow these steps:

- draws the $\tilde{\mathbf{x}}$, $\tilde{\mathbf{m}}$, and $\tilde{\mathbf{n}}$ samples, assembles $\tilde{\mathbf{z}}$ with them (Equation 2) and computes $\bar{\mathbf{x}}$, which is the output of the generator G (Equation 3);
- optimizes the discriminator (in line 14) with the loss function (line 15) given by Expression 4;
- optimizes the generator (in line 16) with the loss function (line 17) given by Expression 5;
- repeats these steps for the given number of iterations (n_iter);

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_10

- after training SGAIN, estimates the whole set of missing values (in lines 19, 20 and 21).

$$\nabla_{\theta_D} \frac{1}{mb} \sum_{j=1}^{mb} \left[\tilde{\mathbf{m}}(j) \odot D\Big(\tilde{\mathbf{x}}(j)\Big) \right] - \frac{1}{mb} \sum_{j=1}^{mb} \left[\Big(1 \ominus \tilde{\mathbf{m}}(j)\Big) \odot D\Big(\bar{\mathbf{x}}(j)\Big) \right] \tag{4}$$

$$\nabla_{\theta_G} - \frac{1}{mb} \sum_{j=1}^{mb} \left[\Big(1 \ominus \tilde{\mathbf{m}}(j)\Big) \odot D\Big(\bar{\mathbf{x}}(j)\Big) \right] + \frac{\alpha}{mb} \sum_{j=1}^{mb} \left[\tilde{\mathbf{m}}(j) \odot \Big(\tilde{\mathbf{x}}(j) \ominus \bar{\mathbf{x}}(j)\Big)^2 \right] \tag{5}$$

For the sake of brevity, the parameter that allows to select a specific optimizer (the options are: Adam, RMSProp, and SGD) as well as the hyper-parameters of that optimizer, are elided in Algorithm 1.

3.3 Wasserstein Slim GAIN with Clipping Penalty

This Section introduces the *Wasserstein Slim GAIN with Clipping Penalty* imputation method (WSGAIN-CP). This method was inspired in the work described in [8, 4, 24] and it aims to reduce the main caveats that affect a vanilla GAN [8, 24], such as mode-collapse and vanishing gradients [4]. The architecture of WSGAIN-CP remains almost identical to that of SGAIN (Figure 1).

Generator The generator of WSGAIN-CP is identical to the counterpart of SGAIN (Section 3.2); they even share the same loss function (see Expression 6 and Expression 5).

$$\nabla_{\theta_G} - \frac{1}{mb} \sum_{j=1}^{mb} \left[\Big(1 \ominus \tilde{\mathbf{m}}(j)\Big) \odot C\Big(\bar{\mathbf{x}}(j)\Big) \right] + \frac{\alpha}{mb} \sum_{j=1}^{mb} \left[\tilde{\mathbf{m}}(j) \odot \Big(\tilde{\mathbf{x}}(j) \ominus \bar{\mathbf{x}}(j)\Big)^2 \right] \tag{6}$$

Critic In a Wasserstein GAN, as it is WSGAIN-CP, the discriminator is named *critic* (C) and is trained more times than the generator. A common configuration is to train the critic 5 times more per each train of the generator. Moreover, for each train of the critic its weights are kept within a predefined interval (usually, that interval is $[-0.01, +0.01]$), a technique known as *(weight) clipping penalty*. It deserves to be noticed that, this penalty is not a component of the loss function of the critic, as shown in Expression 7 (which is identical to Expression 4) and in Lines 18 and 19 of Algorithm 2.

$$\nabla_{\theta_C} \frac{1}{mb} \sum_{j=1}^{mb} \left[\tilde{\mathbf{m}}(j) \odot C\Big(\tilde{\mathbf{x}}(j)\Big) \right] - \frac{1}{mb} \sum_{j=1}^{mb} \left[\Big(1 \ominus \tilde{\mathbf{m}}(j)\Big) \odot C\Big(\bar{\mathbf{x}}(j)\Big) \right] \tag{7}$$

WSGAIN-CP Algorithm For the sake of brevity, we elide to fully describe the WSGAIN-CP algorithm (Algorithm 2). However, one should notice that the loss function of the critic (in line 18) is given by Expression 7, whereas the loss function of the generator (in line 22) is given by Expression 6. In line 19, the weights of the critic network are kept within an interval whose bounds are determined by the *clip* parameter. Again, the parameter that allows to select a specific optimizer (the options are: Adam, RMSProp, and SGD), as well as the hyper-parameters of that optimizer, are elided in Algorithm 2.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_10

Algorithm 2: Pseudo-code of WSGAIN-CP

Input: $\mathbf{X}$ // dataset w/ missing values
Parameter: mb // mini-batch size
Parameter: α // hyper-parameter of Generator loss
Parameter: clip // clip value of Critic weights
Parameter: n_iter // number of iterations
Parameter: n_critic // additional times to train the Critic
Result: $\hat{\mathbf{X}}$ // imputed dataset
$\mathbf{M} \leftarrow mask(\mathbf{X})$ // each miss. value is 0, otherwise 1
for *iter* $\leftarrow 1$ *to* *n_iter* **do**
  for *extra* $\leftarrow 1$ *to* *n_critic* **do**
    Draw *mb* samples from $\mathbf{X}$: $\{\tilde{\mathbf{x}}(j)\}_{j=1}^{mb}$
    Draw *mb* samples from $\mathbf{M}$: $\{\tilde{\mathbf{m}}(j)\}_{j=1}^{mb}$
    Draw *mb* i.i.d. samples of $\mathbf{N}$: $\{\tilde{\mathbf{n}}(j)\}_{j=1}^{mb}$ // random noise
    for $j \leftarrow 1$ *to* *mb* **do**
      $\tilde{\mathbf{z}}(j) \leftarrow \tilde{\mathbf{m}}(j) \odot \tilde{\mathbf{x}}(j) \oplus \left(1 \ominus \tilde{\mathbf{m}}(j)\right) \odot \mathbf{n}(j)$ // Equation 2
      $\bar{\mathbf{x}}(j) \leftarrow G\left(\tilde{\mathbf{z}}(j), \tilde{\mathbf{m}}(j)\right)$ // Equation 3
    end
    // (1) Critic Optimization
    Update C using Adam or RMSprop or SGD
    $\nabla_{\theta_C} \frac{1}{mb} \sum_{j=1}^{mb} \mathcal{L}_C \left(C\left(\tilde{\mathbf{x}}(j)\right), C\left(\bar{\mathbf{x}}(j)\right), \tilde{\mathbf{m}}(j) \right)$
    $w_c \leftarrow clip\_critic\_weights(w_c, -clip, +clip)$
  end
  // (2) Generator Optimization
  Update G using Adam or RMSprop or SGD
  $\nabla_{\theta_G} - \frac{1}{mb} \sum_{j=1}^{mb} \mathcal{L}_G \left(C\left(\bar{\mathbf{x}}(j), \tilde{\mathbf{m}}(j)\right) \right) + \frac{\alpha}{mb} \sum_{j=1}^{mb} \mathcal{L}_{MSE}\left(\tilde{\mathbf{x}}(j), \bar{\mathbf{x}}(j), \tilde{\mathbf{m}}(j)\right)$
end
$\mathbf{Z} \leftarrow \mathbf{M} \odot \mathbf{X} \oplus (1 \ominus \mathbf{M}) \odot \mathbf{N}$ // Equation 2
$\bar{\mathbf{X}} \leftarrow G(\mathbf{Z}, \mathbf{M})$ // Equation 3
$\hat{\mathbf{X}} \leftarrow \mathbf{M} \odot \mathbf{X} \oplus (1 \ominus \mathbf{M}) \odot \bar{\mathbf{X}}$ // Equation 1

3.4 Wasserstein Slim GAIN with Gradient Penalty

This section introduces the *Wasserstein Slim GAIN with Gradient Penalty* imputation method (WSGAIN-GP). This method was inspired in the work described in [8, 4, 10, 24] and the motivation behind it is to reduce the main caveats that affect a vanilla GAN [8, 24] as well as the ones that affect a WGAN [4], namely the undesired behaviour that can arise due to weight clipping [10]. The architecture of WSGAIN-GP remains almost identical to that of WSGAIN-CP.

Generator The generator of WSGAIN-GP is identical to the counterpart of WSGAIN-CP (Section 3.3).

Critic The critic (C) of WSGAIN-GP is almost identical to the counterpart of WSGAIN-CP (Section 3.3). However, since WSGAIN-GP aims to get rid of the undesired behaviour that can arise due to weight clipping [10], there is no weight clipping. Instead, to improve its training the critic uses a technique known as

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_10

gradient penalty. The gradient penalty is a component of the loss function, which is the only difference between Expression 7 and Expression 9.

Algorithm 3: Pseudo-code of WSGAIN-GP

Input: $\mathbf{X}$ // dataset w/ missing values
Parameter: mb // mini-batch size
Parameter: α // hyper-parameter of Generator loss
Parameter: *lambda* // hyper-parameter of Critic loss
Parameter: n_iter // number of iterations
Parameter: n_critic // additional times to train the Critic
Result: $\hat{\mathbf{X}}$ // imputed dataset
$\mathbf{M} \leftarrow mask(\mathbf{X})$ // each miss. value is 0, otherwise 1
for *iter* $\leftarrow 1$ *to* *n_iter* **do**
  for *extra* $\leftarrow 1$ *to* *n_critic* **do**
    Draw *mb* samples from $\mathbf{X}$: $\{\tilde{\mathbf{x}}(j)\}_{j=1}^{mb}$
    Draw *mb* samples from $\mathbf{M}$: $\{\tilde{\mathbf{m}}(j)\}_{j=1}^{mb}$
    Draw *mb* i.i.d. samples of $\mathbf{N}$: $\{\tilde{\mathbf{n}}(j)\}_{j=1}^{mb}$ // random noise
    Draw *mb* i.i.d. samples of $\mathbf{N}$: $\{\tilde{\epsilon}(j)\}_{j=1}^{mb}$ // random noise
    for $j \leftarrow 1$ *to* *mb* **do**
      $\tilde{\mathbf{z}}(j) \leftarrow \tilde{\mathbf{m}}(j) \odot \tilde{\mathbf{x}}(j) \oplus \Big(1 \ominus \tilde{\mathbf{m}}(j)\Big) \odot \mathbf{n}(j)$ // Equation 2
      $\bar{\mathbf{x}}(j) \leftarrow G\Big(\tilde{\mathbf{z}}(j), \tilde{\mathbf{m}}(j)\Big)$ // Equation 3
      $\dot{\mathbf{x}}(j) \leftarrow \tilde{\mathbf{m}}(j) \odot \Big(\tilde{\epsilon}(j) \odot \tilde{\mathbf{x}}(j)\Big) \oplus \Big(\Big(1 \ominus \tilde{\mathbf{m}}(j)\Big) \odot \Big(1 \ominus \tilde{\epsilon}(j)\Big) \odot \bar{\mathbf{x}}(j)\Big)$
    end
    // (1) Critic Optimization
    Update C using Adam or RMSprop or SGD
    $\nabla_{\theta_C} \frac{1}{mb} \sum_{j=1}^{mb} \mathcal{L}_C\Big(C\Big(\tilde{\mathbf{x}}(j)\Big), C\Big(\bar{\mathbf{x}}(j)\Big), \tilde{\mathbf{m}}(j)\Big) + \frac{\lambda}{mb} \sum_{j=1}^{mb} \mathcal{L}_{GradPen}\Big(C\Big(\dot{\mathbf{x}}(j)\Big)\Big)$
  end
  // (2) Generator Optimization
  Update G using Adam or RMSprop or SGD
  $\nabla_{\theta_G} - \frac{1}{mb} \sum_{j=1}^{mb} \mathcal{L}_G\Big(C\Big(\bar{\mathbf{x}}(j), \tilde{\mathbf{m}}(j)\Big)\Big) + \frac{\alpha}{mb} \sum_{j=1}^{mb} \mathcal{L}_{MSE}\Big(\tilde{\mathbf{x}}(j), \bar{\mathbf{x}}(j), \tilde{\mathbf{m}}(j)\Big)$
end
$\mathbf{Z} \leftarrow \mathbf{M} \odot \mathbf{X} \oplus (1 \ominus \mathbf{M}) \odot \mathbf{N}$ // Equation 2
$\bar{\mathbf{X}} \leftarrow G(\mathbf{Z}, \mathbf{M})$ // Equation 3
$\hat{\mathbf{X}} \leftarrow \mathbf{M} \odot \mathbf{X} \oplus (1 \ominus \mathbf{M}) \odot \bar{\mathbf{X}}$ // Equation 1

$$\dot{\mathbf{x}}(j) = \tilde{\mathbf{m}}(j) \odot \Big(\tilde{\epsilon}(j) \odot \tilde{\mathbf{x}}(j)\Big) \oplus \Big(\Big(1 \ominus \tilde{\mathbf{m}}(j)\Big) \odot \Big(1 \ominus \tilde{\epsilon}(j)\Big) \odot \bar{\mathbf{x}}(j)\Big) \tag{8}$$

$$\nabla_{\theta_C} \frac{1}{mb} \sum_{j=1}^{mb} \Big[\tilde{\mathbf{m}}(j) \odot C\Big(\tilde{\mathbf{x}}(j)\Big)\Big] - \frac{1}{mb} \sum_{j=1}^{mb} \Big[\Big(1 \ominus \tilde{\mathbf{m}}(j)\Big) \odot C\Big(\bar{\mathbf{x}}(j)\Big)\Big] + \frac{\lambda}{mb} \sum_{j=1}^{mb} \Big(\Big\|\nabla_{\dot{\mathbf{x}}(j)} C\Big(\dot{\mathbf{x}}(j)\Big)\Big\|_2 \ominus 1\Big)^2 \tag{9}$$

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_10

WSGAIN-GP Algorithm For the sake of brevity, we elide to fully describe the WSGAIN-GP algorithm (Algorithm 3). However, four details deserve to be noticed: (i) the new random noise (line 13) is generated as described in Section 3.2; (ii) the statement in line 17 corresponds to Equation 8 and is crucial to compute the norm of the gradients (line 20 and Expression 9); (iii) the loss function of the critic (line 20) is given by Expression 9; as aforementioned, the gradient penalty component is the only difference between Expression 7 and Expression 9; and (iv) the loss function of the generator (line 23) is exactly the same of that in WSGAIN-CP Algorithm 2 (line 22) and is given by Expression 6. Again, the parameter that allows to select a specific optimizer (the options are: Adam, RMSProp, and SGD) as well as the hyper-parameters of that optimizer, are elided in Algorithm 3.

4 Experimental Results

This section evaluates the novel generative imputation methods on 10 real-world datasets from the ML repository maintained by the University of California at Irvine [6], as presented in Table 1.

Table 1: Short description of datasets.

Name	Area	Instances	Attributes	Continuous	Discrete	Target	Model
Breast Cancer	Life	569	31 (32)	30	0	Diagnosis	LR
Credit Card	Business	30000	24 (25)	14	9	Def. Pay. Next Month	LR
EEG Eye State	Life	14980	15 (15)	14	0	Eye Detect.	KNN
Iris	Life	150	5 (5)	4	0	Class	KNN
Letter Recognition	Computer	20000	17 (17)	16	0	Letter	KNN
Online News Popularity	Business	39644	60 (61)	56	3	Shares	LR
Spambase	Computer	4601	58 (58)	57	0	Spam	LR
(Red) Wine Quality	Business	1599	12 (12)	11	0	Quality	KNN
(White) Wine Quality	Business	4898	12 (12)	11	0	Quality	KNN
Yeast	Life	1484	9 (10)	6 (7)	2 (1)	Local. Site	KNN

During the data preprocessing several data transformations were applied: drop one (non-relevant) attribute (aka variable or feature) in Breast Cancer, Credit Card, Online News Popularity, and Yeast datasets; perform one-hot encoding of discrete (aka categorical) attributes; and scale the continuous (aka numerical) attributes to fit inside the interval $[-1.00, +1.00]$, using the scikit-learn [16] `MinMaxScaler`[5]. In the Yeast dataset we considered one continuous

[5] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_10

attribute as being discrete, thus, it has two discrete attributes instead of just one. The missing values are introduced under an MCAR setting and, unless otherwise stated, the amount of missing values is 20% of all data points. Moreover, the amputation is evenly distributed by all attributes. However, it is highly likely for an amputated dataset to have rows with just missing values, we did nothing to prevent such cases.

We obtained GAIN from its GitHub repository[6] and for every run we kept unchanged the batch size (128), the hint rate (0.9), the hyper-parameter alpha (100), and the number of iterations (9000). To promote a fair comparison, we also used these values to run SGAIN, WSGAIN-CP, and WSGAIN-GP (we recall that our methods do not have the hint generator). However, since the critic of WSGAIN-CP and WSGAIN-GP was trained five times more than the generator, we decided to divide the number of iterations by three, in these cases. For WSGAIN-CP and WSGAIN-GP the generator was trained 3000 times, whereas the critic was 15000 times, which sums up to as many times the generator and the discriminator of GAIN are trained (each 9000 times).

Three types of experimental results are discussed in the next sections: (i) response times, which measure how fast the methods can present results; (ii) the quality of the results, measured by the root mean square error (RMSE) between the imputed values and the original deleted values; and (iii) the area under the receiver operating characteristics (AUROC), which in this study is used to measure a model accuracy of post-imputation prediction.

4.1 Response Times

Figure 2 shows the mean execution time, taken from ten executions, of GAIN, SGAIN, WSGAIN-CP and WSGAIN-GP on each dataset. The exact same amputated datasets were used by each algorithm to perform the imputation of missing values. In all cases, SGAIN outperformed GAIN, with improvements ranging from (roughly) 20 to 30% less than the execution time of GAIN. WSGAIN-CP and WSGAIN-GP also outperformed GAIN, the only exception is that GAIN is marginally faster than WSGAIN-CP on the Letter Recognition dataset. In general, SGAIN, WSGAIN-CP and WSGAIN-GP are considerable faster than GAIN, in particularly SGAIN. This is an extremely significant result since each one of our novel generative imputation methods are trained faster than GAIN.

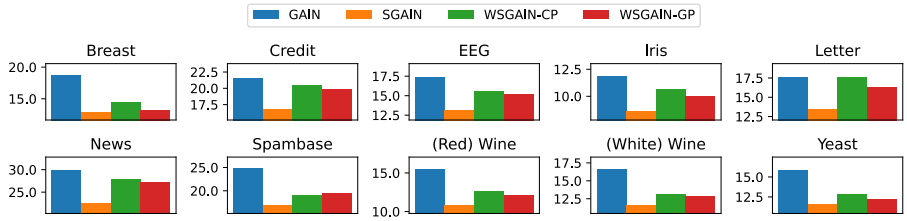

Fig. 2: Response times in seconds (lower is better).

[6] https://github.com/jsyoon0823/GAIN

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_10

4.2 RMSE Performance

A common way to compare the quality of the imputed data is to measure how close are the imputed data points to the counterpart data points in the original (i.e., complete) dataset. Usually, this is achieved by computing the *root mean square error* (RMSE). Figure 3 reports on the RMSE achieved by GAIN, SGAIN, WSGAIN-CP, and WSGAIN-GP on all datasets and from ten executions. The exact same amputated datasets were used by each algorithm to perform the imputation of missing values.

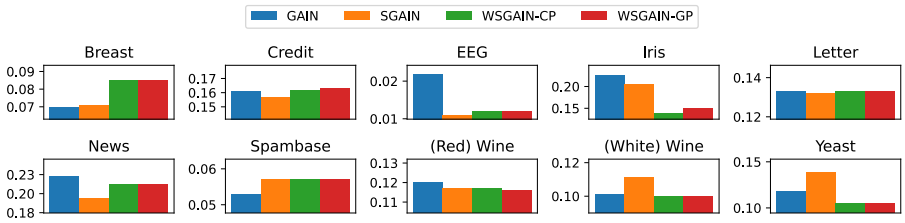

Fig. 3: RMSE performance (values range from 0 to 1, lower is better).

The main observation that can be derived from the RMSE results is that our novel generative imputation methods have competitive performance to that of GAIN. This is again a very significant result, since it shows, besides the observed consistency, that the changes and optimizations of the architecture of SGAIN, from which WSGAIN-CP and WSGAIN-GP were derived, do not impair the quality of the missing data imputation.

4.3 AUROC Performance

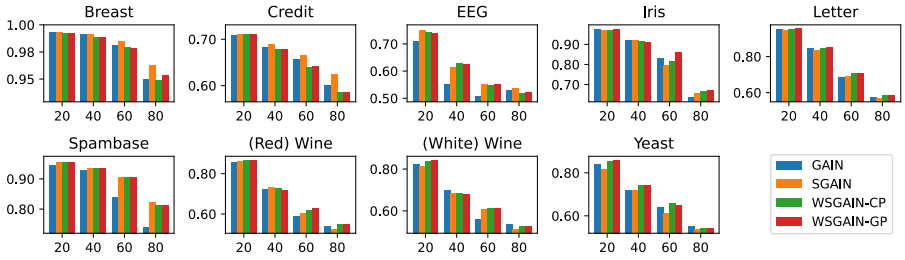

Fig. 4: AUROC performance (values range from 0 to 1, higher is better); the amounts of missing values are 20%, 40%, 60%, and 80%.

In this study, the areas under the receiver operating characteristics (AUROC) are used to measure the models accuracies of post-imputation predictions. The models are the *Logistic Regression* (LR)[7] and the K Neighbours Classifier

[7] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_10

(KNN)[8] (see rightmost column of Table 1). We do not present results for the (Online) News (Popularity) dataset since the post-imputation predictions took too longer to complete due to the size of the dataset.

The main observation that can be derived from the AUROC results shown in Figure 4 is that both new WSGAIN imputation methods present the higher values in almost all datasets and for all settings of missing values. This is again a very significant result, since it shows, besides the observed consistency, that the changes and optimizations of the architecture of SGAIN, from which WSGAIN-CP and WSGAIN-GP were derived, do not impair the models accuracies of post-imputation predictions.

5 Conclusions

This paper presented and discussed three novel generative imputation methods: SGAIN, WSGAIN-CP, and WSGAIN-GP. SGAIN is a slimmer GAN version, whereas WSGAIN-CP and WSGAIN-GP are variations of a Wasserstein GAN. These methods are available online in a GitHub repository[9].

The performed experimental work comparatively evaluated our novel methods with the competition, using real-world datasets from different domains, with distinct characteristics and under various settings. Results explicitly showed that our methods outperformed the reference GAN-based imputation method (GAIN) and implicitly showed that they outperformed other imputation methods (e.g. MICE, MissForest, Matrix Completion, Auto-Encoder, and EM) [24, 7]. This latter conclusion is derived from the fact that our methods outperform GAIN and GAIN outperforms those data imputation methods, as shown in [24]. The measured response times also showed that the newly developed methods are considerably faster than GAIN and require significant less time to be trained.

6 Acknowledgements

The first author thanks the ALGORITMI research centre, Universidade do Minho, where he conducts part of his research as an external collaborator. The third author developed his work at ALGORITMI research centre, Universidade do Minho, supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

References

1. Abedjan, Z., Chu, X., Deng, D., Fernandez, R.C., Ilyas, I.F., Ouzzani, M., Papotti, P., Stonebraker, M., Tang, N.: Detecting data errors: Where are we and what needs to be done? Proceedings of the VLDB Endowment **9**(12), 993–1004 (2016)
2. Abedjan, Z., Golab, L., Naumann, F.: Profiling relational data: a survey. The VLDB Journal **24**(4), 557–581 (2015)
3. Arjovsky, M., Bottou, L.: Towards principled methods for training generative adversarial networks. arXiv preprint arXiv:1701.04862 (2017)

[8] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[9] https://github.com/dtneves/ICCS_2021

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_10

4. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein gan. arXiv preprint arXiv:1701.07875 (2017)
5. Buuren, S.v., Groothuis-Oudshoorn, K.: mice: Multivariate imputation by chained equations in r. Journal of statistical software pp. 1–68 (2010)
6. Dua, D., Graff, C.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml
7. Friedjungová, M., Vašata, D., Balatsko, M., Jiřina, M.: Missing features reconstruction using a wasserstein generative adversarial imputation network. In: International Conference on Computational Science. pp. 225–239. Springer (2020)
8. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. Advances in neural information processing systems **27**, 2672–2680 (2014)
9. Graham, J.W.: Missing data analysis: Making it work in the real world. Annual review of psychology **60**, 549–576 (2009)
10. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of wasserstein gans. In: Advances in neural information processing systems. pp. 5767–5777 (2017)
11. Huqqani, A.A., Schikuta, E., Ye, S., Chen, P.: Multicore and gpu parallelization of neural networks for face recognition pp. 349–358 (2013)
12. Lall, R.: How multiple imputation makes a difference. Political Analysis **24**(4), 414–433 (2016)
13. LeCun, Y.A., Bottou, L., Orr, G.B., Müller, K.R.: Efficient backprop. In: Neural networks: Tricks of the trade, pp. 9–48. Springer (1998)
14. Liefeldt, L., Brakemeier, S., Glander, P., Waiser, J., Lachmann, N., Schönemann, C., Zukunft, B., Illigens, P., Schmidt, D., Wu, K., et al.: Donor-specific hla antibodies in a cohort comparing everolimus with cyclosporine after kidney transplantation. American journal of transplantation **12**(5), 1192–1198 (2012)
15. Little, R.J., Rubin, D.B.: Statistical analysis with missing data, vol. 793. John Wiley & Sons (2019)
16. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. the Journal of machine Learning research **12**, 2825–2830 (2011)
17. Rubin, D.B.: Multiple imputation for nonresponse in surveys, vol. 81. John Wiley & Sons (2004)
18. Schafer, J.L., Graham, J.W.: Missing data: our view of the state of the art. Psychological methods **7**(2), 147 (2002)
19. Stekhoven, D.J., Bühlmann, P.: Missforest—non-parametric missing value imputation for mixed-type data. Bioinformatics **28**(1), 112–118 (2012)
20. Strigl, D., Kofler, K., Podlipnig, S.: Performance and scalability of gpu-based convolutional neural networks. In: 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing. pp. 317–324. IEEE (2010)
21. Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., Altman, R.B.: Missing value estimation methods for DNA microarrays . Bioinformatics **17**(6), 520–525 (06 2001), https://doi.org/10.1093/bioinformatics/17.6.520
22. Van Buuren, S.: Flexible imputation of missing data. CRC press (2018)
23. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: Openml: Networked science in machine learning. SIGKDD Explorations **15**(2), 49–60 (2013), http://doi.acm.org/10.1145/2641190.2641198
24. Yoon, J., Jordon, J., Van Der Schaar, M.: Gain: Missing data imputation using generative adversarial nets. arXiv preprint arXiv:1806.02920 (2018)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_10