Predicting Online Order Satisfaction with Lagged Data Using WGAIN-GP

Bette Donker[a], Evita Hoogeveen[a], Lars Hurkmans[a], Daan Schopmeijer[a], Flavius Frasincar[a,*], Enzo Ido[a], Jasmijn Klinkhamer[a]

[a]*Erasmus University Rotterdam, P.O. Box 1738, NL-3000 DR Rotterdam, the Netherlands*

Abstract

E-commerce has grown a lot in recent years and so has the research performed in this field. In this paper we estimate order satisfaction by predicting outcomes of relevant variables at the moment an order is made, such that companies can act on this signal. In order to deal with data that is not known at the order date (i.e., lagged missing data), we propose an extension of an existing generative imputation method. The Generative Adversarial Imputation Network (GAIN) is suitable for data imputation on tabular datasets. A more stable method is the Wasserstein GAIN (WGAIN). In this paper, we propose to improve this method by adding the Gradient Penalty to WGAIN resulting in WGAIN-GP. We perform experiments on a large dataset from a Dutch online retailer. Using WGAIN-GP we obtain a better accuracy of 61% at the order date compared to 54% and 53% obtained by GAIN and WGAIN, respectively.

Keywords: Generative Adversarial Imputation Network, Data Imputation, Decision Tree

1. Introduction

Nowadays, more and more people buy products online. Online retailers feel the need to predict order satisfaction in order to keep customers happy. For example, [1] investigated how product attributes, average consumer ratings, and single affect-rich positive or negative consumer reviews influence the online purchasing decisions of younger and older adults. In these online shops the online retailer sometimes functions as an intermediary. This means order satisfaction can be greatly dependent on the product supplier (i.e., the partner). Measuring order satisfaction can be summarized in an overview of order satisfaction per partner. This allows an online retailer to quickly respond to partners whose performance is not good enough. The sooner order satisfaction can be predicted, the earlier this response can be. Furthermore, predicting order satisfaction can help to foresee possible issues during the order execution. Once again, the faster the prediction, the earlier the company can act upon this.

*Corresponding author; tel: +31 (0)10 408 1262

Email addresses: 408249bd@eur.nl (Bette Donker), 426768eh@eur.nl (Evita Hoogeveen), 453028lh@eur.nl (Lars Hurkmans), 503403ds@eur.nl (Daan Schopmeijer), frasincar@ese.eur.nl (Flavius Frasincar), 534787ei@eur.nl (Enzo Ido), 584059jk@eur.nl (Jasmijn Klinkhamer)

In this paper, we aim to predict order satisfaction based on order data from a Dutch online retailer. This can be treated as a multi-class classification problem. In order to devise a method to find the correct match labels, the choice for decision trees seems logical. An advantage of decision trees is that they have a high interpretability. However, a disadvantage is that they have a high variability, which means a single decision tree might yield very inaccurate results. For this reason, usually single decision trees are combined to improve the accuracy, and this technique is classified as an ensemble method [2].

However, the problem we face can also be seen from another perspective. In our case, we do not consider the class labels directly, but we rather try to infer the labels from the characteristics of some underlying variables, which we call leading variables. Once the outcomes of these leading variables are known, classification can be made with certainty. Therefore, the uncertainty of predicting lies in correctly predicting the outcomes of these leading variables. As the information of the leading variables only becomes available later in time, waiting for this information leads to accurate predictions. This has one main disadvantage, namely, the relevance of the predictions decreases. As relevance is of great concern to us, we therefore cannot wait until the outcomes of the leading variables become available. For this reason we aim to predict (impute) outcomes of the leading variables as soon as possible, ideally at the moment at which the order takes place. This leads us to the following research question:

How to accurately predict outcomes of leading variables immediately after the order date?

Predicting outcomes of the leading variables immediately raises the following sub-question:

How can we treat data that is not known at the order date, but we know will come available later on in time?

To answer these questions, we need to refine our understanding of what predicting the outcomes of leading variables exactly entails. Since our aim is to predict the outcomes of leading variables as soon as possible, and we miss the data for some period in time, we can treat the outcomes as missing data. By taking this perspective, we can therefore argue that we try to impute missing data. Hence, we look for methods which are able to impute missing data. For this, it is important to know the kind of data that is missing when there are no constraints. There are three types of missing data [3]. The first one is data Missing Completely At Random (MCAR). Second, there can be data Missing At Random (MAR). This is the case when missingness solely depends on the observed variables. Third, data can be Missing Not At Random (MNAR) when the missingness depends on the observed and unobserved variables. In this paper, we assume that our data is MAR. That is, we assume that we can predict outcomes of our leading variables by looking at data available at the moment the order is made.

Given that we assume our missing data is MAR, we can narrow our search for methods which are able to deal with data that is MAR. Data imputation is an important topic in machine learning [4, 5, 6, 7]. For this, we can consider both discriminative (e.g., MICE [8, 9] and k-NN [10]) and generative methods (e.g., VAE [11] and GAN [12]). However, research concentrating on generative methods is most recent [13, 14, 15] and has been shown to produce better results than the ones concentrating on discriminative methods [3]. We define deep generative models as multilayer neural networks that generate samples corresponding to the data distribution. Deep generative models can be divided in two groups: Variational Autoencoder (VAE) [11] and Generative Adversarial Network (GAN) methods [12]. The most important difference between the two is that the VAE aims to explicitly model the data distribution, while the GAN tries to implicitly model the data distribution. VAEs allow for very specific feature selection on generated data and they allow modifying existing data. For this reason they have been shown to be successful in syntatic Web Service discovery [16]. [17] describes a method using VAEs as an alternative for GAN in data imputation. Despite the fact that GANs are harder to train than VAEs they usually generate more realistic samples with a higher level of variability. By modelling the data implicitly as in GANs, we overcome some problems that arise when explicitly modelling the data. An example of such a problem is that in maximum likelihood estimation with complex functions as in VAE, it can occur that we approximate intractable probabilistic expressions [12]. Next to this advantage over VAEs, GANs have been evolving ever since they were introduced in 2014. They have shown their contribution in areas of image generation and text generation [18]. In the past, the GAN framework also has been translated to work well with tabular data, the type of data we are dealing with in this paper. We illustrate this with an experiment with an Intensive Care Unit dataset done by [19]. Here, GANs are used to figure out the distribution of a multivariate time series dataset where the GAN outputs values that were missing for some samples. The goal in this experiment is to quickly find missing values that could help predict whether the patient dies in a hospital.

In this paper we focus on data imputation and therefore we do not use the GAN (regular) framework, but the GAN imputation framework as in [3]. As a result the GAN is renamed to a Generative Adverserial Imputation Network (GAIN). GAIN has been evaluated in multiple studies against other state-of-the-art methods often providing better results [20, 21, 22, 23, 24]. Not all leading variables in our dataset are suitable for GAIN. Depending on the distribution of the leading variables, we also consider reducing leading variables to binary variables. In case the leading variable has a highly imbalanced distribution, where one outcome has a very high probability, the use of a GAIN framework is too complicated. For this reason, we propose an alternative imputation method for these instances in the form of multinomial sampling. Multinomial sampling is a simple method, as it does not take any characteristics of an order into account. However, its usage can be justified if it helps us know with very high certainty (say 96% or higher) what

the outcome is beforehand. In these circumstances it is questionable if the usage of more complex methods, such as the GAIN framework will add any real value, given the computational time it takes to train these methods.

For the variables which we intend to impute with the use of the GAIN framework, we must note that in the literature it is shown that the original GAN and GAIN frameworks might be too unstable to train [25]. Therefore, we also use an alternative method: Wasserstein GAIN (WGAIN). WGAIN is more stable compared to GAIN because it uses an alternative cost function to solve the issue of a vanishing gradient that restricts parts of the GAIN to work properly. Besides this, we propose an improved version of WGAIN called the WGAIN-GP. The WGAIN-GP enhances the WGAIN method by using a gradient penalty instead of weight clipping to make sure constraints are met. With a large clipping parameter it can take long for weights to reach their optimum. Using the penalty to create a flexible boundary reduces training time needed to obtain good results.

Although we compare three generative imputation methods (GAIN, WGAIN, and WGAIN-GP), we cannot conclude how well generative methods as a class perform in our setting. For this reason, we also propose a baseline method. This baseline is based on predicting outcomes using multinomial sampling, for which the empirical distributions of the data are used. The reason to include this baseline is merely to see if more advanced methods are able to outperform a very simple model.

The main contributions of this paper are as follows. Our first contribution is adding the gradient penalty to the WGAIN method. Our second contribution is that we focus on lagged missing data and not on regular missing data. To our knowledge this type of data has not been previously investigated with a generative imputation method. Third, and, last we apply the proposed method to predict customer satisfaction for orders, which is unique in the literature. The developed code in Python is available from `https://github.com/DaanSc/wgain-gp`.

This paper is structured as follows. In Section 2, we review the relevant literature. Afterwards, in Section 3, we describe our proposed method WGAIN-GP. Then, in Section 4, we discuss the data used in our research. After that, in Section 5, we explain the methodology needed for using WGAIN-GP and multinomial sampling in our problem. Subsequently in Section 6, we compare and evaluate the results. Finally, in Section 7, we summarize the key findings and state further recommendations. We give an overview of all the terms and definitions in this paper in Table 1.

Term	Definition
AUC	Area Under the Curve
Ceteris Paribus Testing	Only changing one hyperparameter, while keeping all others fixed.
Distributional Resemblance	How well the distribution of the imputed values of a variable matches the true empirical distribution of the actual data. This can be represented in terms of histograms and Precision-Recall Distribution plots.
GAN	Generative Adversarial Network
GAIN	Generative Adversarial Imputation Network
Lagged Data	Data that is currently missing, but known at a later point in time.
Leading Variables	A group of variables which concern the cancellation, delivery date, return, and case of an order. Knowing the outcomes of these variables implies knowledge about the match labels.
Marginal Contribution	The moment in time between the order date and 30 days after the order date in which a leading variable is able to change the match label of an order.
MAR	Missing At Random
MCAR	Missing Completely At Random
MNAR	Missing Not At Random
Multinomial Sampling	A way to impute missing data based on the observed distributions. For this the binomial/multinomial distribution is used.
Relevance Assumption	The relevance assumption relates to the trade-off between the accuracy and relevance of the prediction of the outcomes of leading variables. When the relevance assumption is stated, we are interested in obtaining predictions immediately after the order date.
RMSE	Root Mean Square Error
RMSPE	Root Mean Square Percentage Error
To be Active	A leading variable is said to be active if it influences the match label of an order.
WGAN	Wasserstein Generative Adversarial Network
WGAN-GP	Wasserstein Generative Adversarial Network with Gradient Penalty
WGAIN	Wasserstein Generative Adversarial Imputation Network
WGAIN-GP	Wasserstein Generative Adversarial Imputation Network with Gradient Penalty

Table 1: Glossary of terms.

2. Related Work

In this section we first discuss previous work on GANs for imputation. After this, we state our contributions to the existing literature. Finally, we elaborate extensively on how we use existing methods to devise our proposed WGAIN-GP method.

2.1. Generative Adversarial Network

In the GAN framework there is both a generative and a discriminative model. The generator aims to generate fake data, of which the distribution closely matches the real data. After this data has been generated, it is passed to the discriminator. The discriminator then has to determine whether the incoming data was generated or not. It does so by comparing how close the generated data resembles the actual data [12]. In the mean time, the generator tries to fool the discriminator by producing fake data that is of 'good' quality. We can think of an example involving policemen and criminals as described in [12]. The policemen can be seen as the discriminator while criminals trying to fabricate fraudulent money can be viewed as the generator. The goal is to eventually create fake instances that resemble the real dataset. Because of the competition that arises in this game, both the policemen and criminals improve their methods until the fake currency is identical to the real currency. There are several GAN frameworks that can handle missing data. The most prominent ones are MisGAN [26], VIGAN [27], CollaGAN [28], and GAIN [3] according to [13]. However, in this paper we will not focus on the first three methods for the following reasons. First, VIGAN is used to address the missing data problem in multi-view data analysis (i.e., different views describe specific context), but we do not have this type of data. Second, GAIN generally outperforms MisGAN for tabular data, which can be due to the reason that MisGAN suffers more from mode collapse [29]. Finally, CollaGAN is used for imputation of missing image data, but we do not review images. We discuss the GAIN framework in the next section.

2.2. Generative Adversarial Imputation Network

Instead of generating samples, like in the GAN framework, our goal is to fill in missing (future) data. A GAIN can be used for this purpose [3]. The GAN principle remains, but the goals of the generator and discriminator are different. The generator first needs to fill in the missing data, after which the discriminator must recognize which data is imputed and which is not. In the GAN framework, the discriminator differentiates the complete sample in terms of real and fake data, while in the GAIN framework, the discriminator receives the entire dataset consisting of parts which are imputed and parts which are not. Therefore, the discriminator does not need to determine whether the data is real or generated, but rather which part of the data was already there and which part is imputed. As a consequence, the input matrix for the discriminator is a combination of real and imputed data, where the imputed data is created from random noise that is passed through the generator. Hence, the discriminator outputs an estimated mask matrix which gives the probability that a value from the real data is observed. To help the discriminator with assigning these probabilities, the GAIN method provides extra information to the discriminator by means of a hint matrix. This hint matrix gives the discriminator partial information about missing data in the original sample and

helps to ensure that the generator actually learns to generate data like the true data distribution. In [3] the missing data is assumed to be MCAR, but the authors showed that GAIN is also working for MAR and MNAR data.

In [23], the performance of the GAIN method is compared with state-of-the-art imputation methods. The authors show that GAIN outperforms the other methods in terms of the Root Mean Squared Error and Fréchet Inception Distance. Moreover, [24] also compared GAIN with other methods and come to the same conclusion that GAIN performs better. Other works [20, 21, 22] also obtained similar results. Figure 1 gives a schematic overview of GAIN.

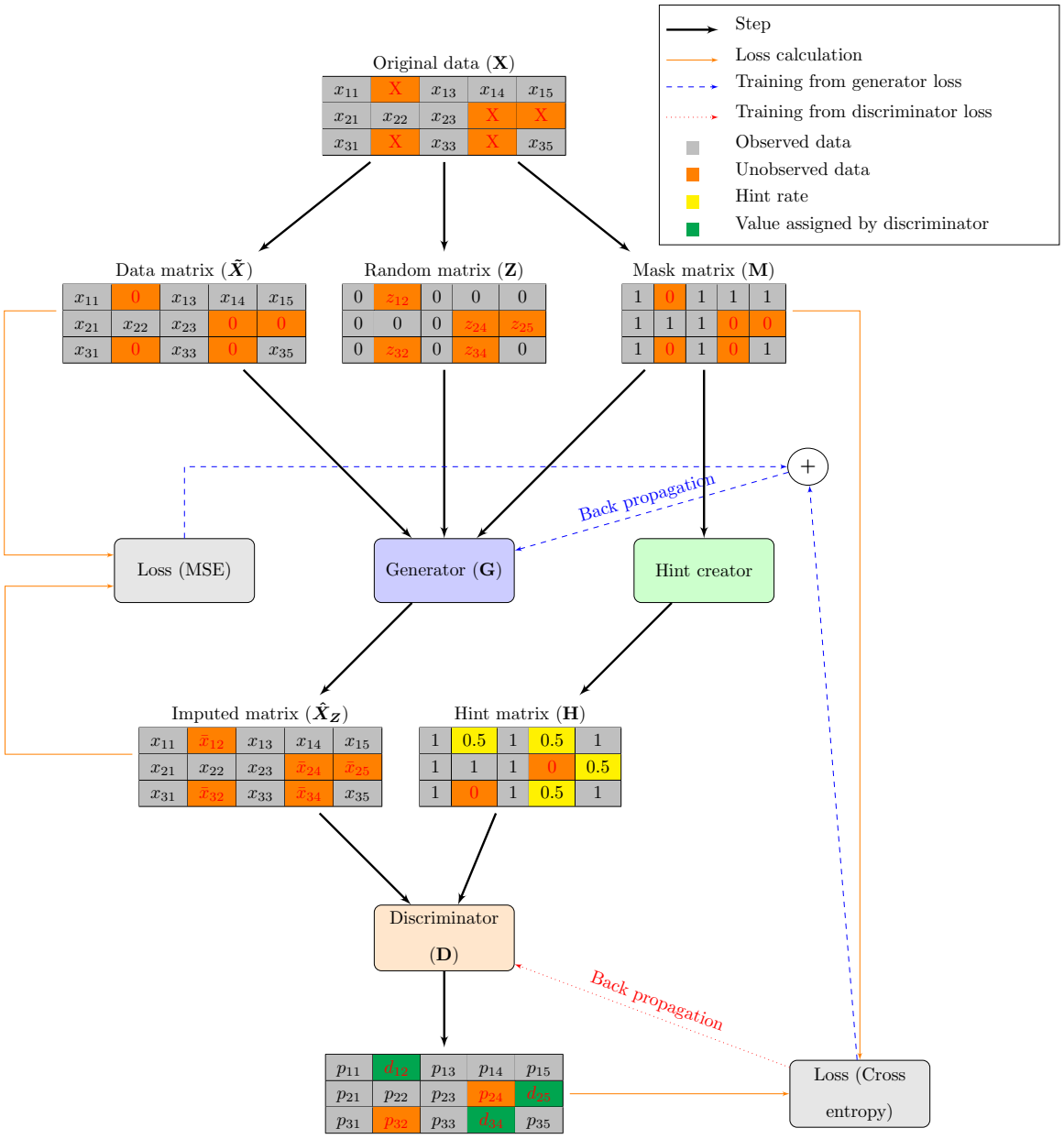


Figure 1: The overall architecture of the GAIN algorithm.

2.3. Wasserstein Generative Adversarial Imputation Network

When using a GA(I)N, several problems can arise [25]. If the generated model distribution q is far away from the real data distribution p, the generator is unable to learn the distribution. When the generator is not working properly, the gradient for the generator diminishes and is not able to provide learning feedback effectively. On the other hand, when the discriminator is optimal, the generator can improve because of the correct information produced by the discriminator. Reason for this problem can be found in the Jensen-Shannon (JS) divergence. This is used in GA(I)Ns to determine the divergence of the two distributions q and p. The metric does not work well when two distributions are not overlapping. When the distributions are not overlapping the same value for the distance is returned and the gradient vanishes [25]. The discriminator will learn to reject this input, but will never get out of this local optimum. Since the input does not change, it provides no new feedback to the generator. The result is that the generator will keep providing the same output. This phenomenon is called mode collapse. As such, the GA(I)N cannot learn and consequently there is no convergence to the real distribution.

To overcome the previous issue of a vanishing gradient, [25] proposes to use an alternative cost function, namely the Wasserstein distance instead of the JS divergence used in GA(I)Ns. By using this Wasserstein distance, the gradient becomes smoother over the whole range. Even when two distributions do not overlap, the Wasserstein distance can still provide a meaningful and smooth representation of the in-between distance, in contrast to the JS divergence metric. The Wasserstein GAN (WGAN) learns regardless of whether the generator is performing well or not. By using this new metric the discriminator does not really classify as imputed or not anymore. The discriminator changes from a classifier to a critic and instead of predicting probabilities, it now predicts scores. These scores correlate to how much the discriminator thinks the samples from the generator are imputed or not. Since the discriminator cannot really discriminate between real and fake quantitative data, we rename the discriminator to critic. For a missing values situation, the aforementioned WGAN can be transferred to WGAIN by combining GAIN with WGAN [13]. WGAIN provides comparable results to GAIN and other well-known imputation methods and outperforms these when there is less than 30% of missing data [13]. Additionally, WGAIN results in more stable training than GAIN. Figure 2 gives a schematic overview of WGAIN.

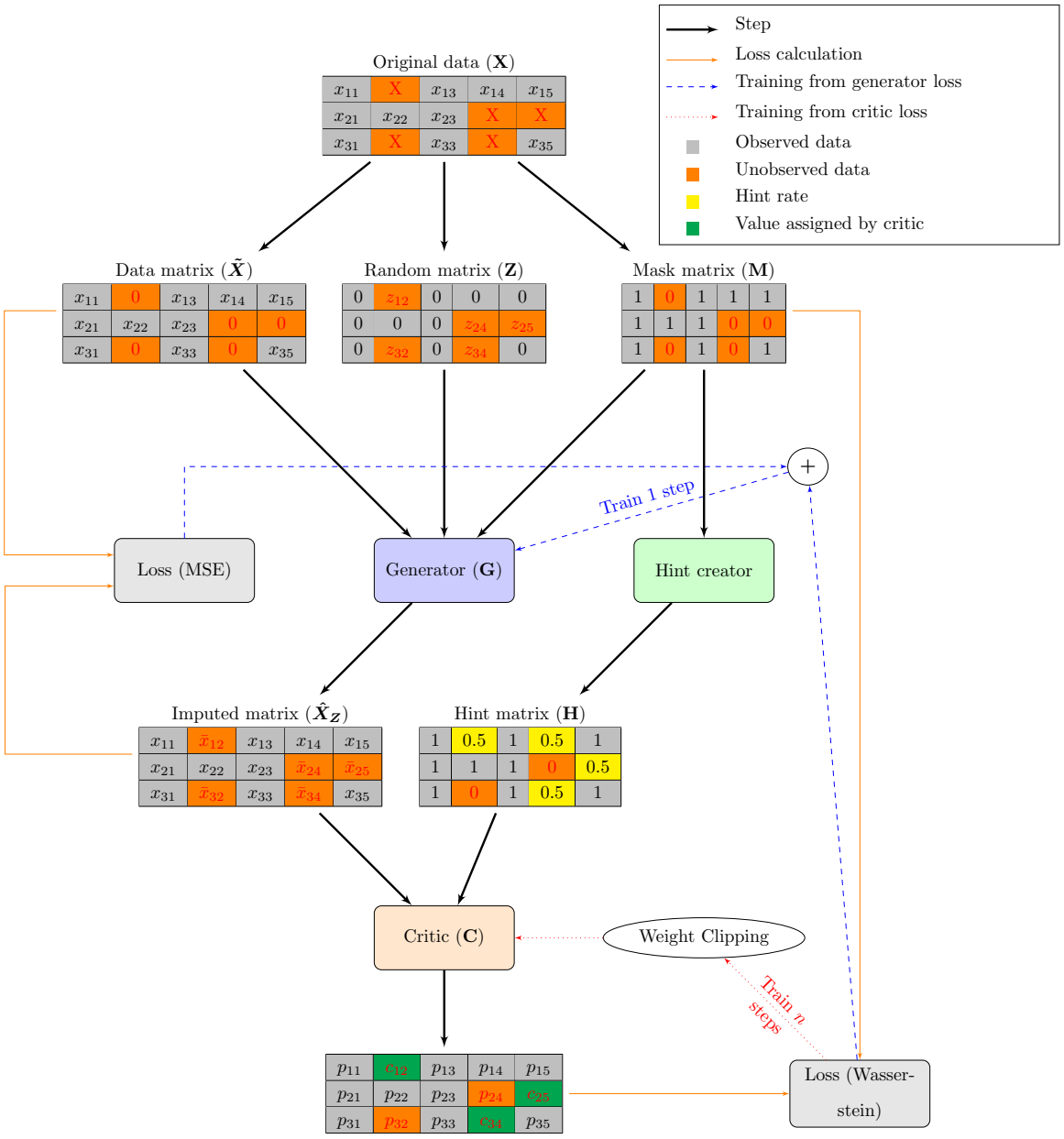


Figure 2: The overall architecture of the WGAIN algorithm.

2.4. Wasserstein Generative Adversarial Network with Gradient Penalty

In order to minimize the WGAN cost function, the slope should not exceed the value 1 in the neighbourhood of the point of interest (i.e., Lipschitz-1 constraint), otherwise the function of the critic is intractable. However, it is difficult to enforce this constraint. In [30], in which the WGAN was proposed, weight clipping was used to enforce the Lipschitz-1 constraint. However, the authors mentioned that weight clipping is not be the most mathematical sound way of enforcing this constraint [30]. When the clipping parameter is large, it can take a long time for any weights to reach their optimum. This makes it harder to train the critic

till optimality. When the clipping parameter is small on the other hand, it can easily lead to vanishing gradients when the number of layers is big or when batch normalization is not used [30]. This problem can be solved by the WGAN-Gradient Penalty (WGAN-GP) which uses a gradient penalty instead of weight clipping to enforce the Lipschitz-1 constraint [30]. The penalty removes the need for clipping and can create a flexible boundary compared to WGAN with clipping. The idea is that WGAN-GP penalizes the model if the gradient norm moves away from its target norm value equal to 1 [30]. This WGAN-GP can also be adjusted to WGAIN-GP in a missing values problem, as discussed in the next section.

3. Wasserstein Generative Adversarial Imputation Network with Gradient Penalty

In this section, we explain thoroughly how to use the previous work on GAIN, WGAIN, and WGAN-GP to construct the WGAIN-GP. Just like the WGAN-GP finds its roots in the WGAN, the WGAIN-GP is a method building on the WGAIN method [13]. Similar to WGAIN, we first denote an $N \times K$ (random) data matrix: $\mathbf{X} = (X_1, ..., X_K)$. Here K corresponds to the number of features in the matrix and $X_1, ..., X_K$ are vectors of size N. Therefore any element of $\mathbf{X}$ can be represented as x_{ij}, for $i = 1, ..., N$ and $j = 1, ..., K$. We say all values in this matrix come from the distribution $P(\mathbf{X})$. After this, we introduce a matrix $\mathbf{M}$, containing the binary values 0 or 1. The matrix $\mathbf{M}$ functions as a mask and provides information about which values we do (not) observe. As such, the distribution of the mask matrix is dependent on the distribution of the missingness present in the data. When M_{ij} is 1, this corresponds to x_{ij} in $\mathbf{X}$ being present. On the other hand, whenever M_{ij} is 0, it aligns with x_{ij} in $\mathbf{X}$ being missing. Furthermore, we introduce a matrix $\tilde{\mathbf{X}}$ that inserts zeros at elements where values in $\mathbf{X}$ are missing,

$$\tilde{\mathbf{X}} = \mathbf{X} \odot \mathbf{M}, \tag{1}$$

in which $\odot$ stands for an element-wise computation. We want to fill in the missing values in $\mathbf{X}$ with the help of data points that are not missing in $\mathbf{M}$ and $\tilde{\mathbf{X}}$. The idea is to make this possible in a generative manner [13]. In order to do so, we establish $\mathbf{Z}$ as an i.i.d. normally distributed ($N(0, \sigma^2)$) matrix with random values, in which we usually take $\sigma^2 = 1$. Consequently, we can define another matrix $\bar{\mathbf{X}}$,

$$\bar{\mathbf{X}} = G(\tilde{\mathbf{X}}, \mathbf{M}, (1 - \mathbf{M}) \odot \mathbf{Z}), \tag{2}$$

where $G(\cdot)$ is the generator function. We then introduce the imputed data matrix,

$$\hat{\mathbf{X}}_{\mathbf{Z}} = \mathbf{X} \odot \mathbf{M} + \bar{\mathbf{X}} \odot (1 - \mathbf{M}). \tag{3}$$

Based on the mask matrix $\mathbf{M}$ we generate a hint matrix $\mathbf{H}$, conditional on the columns we want to impute. The hint matrix can have three values: 0, 0.5, and 1. Whenever an element of the hint matrix equals 0, this implies that the corresponding element in $\mathbf{M}$ also equals 0. Similarly, whenever an element of the hint matrix equals 1, the corresponding element of $\mathbf{M}$ equals 1. If an element of the hint matrix has the value 0.5, the corresponding element in $\mathbf{M}$ can either be 0 or 1. The hint matrix thus gives hints about which elements are imputed and which are not. The value 0.5 is provided to contaminate information, coming from observed and unobserved data in $\mathbf{M}$, about which elements are imputed and which are not. How many elements we want to contaminate depends on the hint rate. A higher hint rate implies less contamination, while a lower hint rate provides more contamination. This hint matrix is sent to the critic in combination with the imputed data matrix $\hat{\mathbf{X}}_{\mathbf{Z}}$.

We can train the critic and the generator by minimizing a loss function. To properly explain how we find the loss function used in the WGAIN-GP method, we have to elaborate on the loss functions used in the GAIN and WGAIN method, first. The GAIN method has as main components the generator and the discriminator instead of the generator and the critic which we use in the WGAIN and WGAIN-GP methods [3]. We train the generator in the GAIN with the help of the standard squared loss function given by Equation 4

$$\mathcal{L}_{MSE}(\hat{\mathbf{X}}_{\mathbf{Z}}, \bar{\mathbf{X}}) = \|\bar{\mathbf{X}} - \hat{\mathbf{X}}_{\mathbf{Z}}\|^2. \tag{4}$$

The motivation of this loss function is as follows. The generator generates data for all elements in $\tilde{\mathbf{X}}$. That is, it generates instances for the data of which we know the actual outcomes as well as for the missing elements. During training, we do not know the values of the missing data, but we do know the values of the data being present. Therefore, the use of this loss function is to force the generator to generate data of the present instances which resemble the present instances as good as possible. Since the data that is missing follows the same distribution as the data that is not missing, the generator also indirectly learns how the values of the missing data can be generated. Thus, this loss function aids in finding the quality of the imputations. However, it does not help us with finding the location of the imputations. Since we also need to know where to impute values, this loss function does not help enough with learning the correct conditional distribution of $\hat{\mathbf{X}}_{\mathbf{Z}}$. One way to get a better performance by the generator, is introducing a discriminator that tries to figure out which instances of $\hat{\mathbf{X}}_{\mathbf{Z}}$ are imputed values and which instances are values coming from the original data $\mathbf{X}$. If the discriminator has a hard time to distinguish the real values from the imputed values that means the generator has done a good job in providing values for the missing variables.

We can extend the previous loss with the Earth Mover's distance (EM distance), also called the Wasserstein distance [13]. This distance has a similar function as the aforementioned loss function, it aims to learn

the conditional distribution of $\hat{\mathbf{X}}_{\mathbf{Z}}$ better. Please note, the WGAIN renames the discriminator to critic, C. Next, we define $C(\hat{\mathbf{X}}_{\mathbf{Z}}, \mathbf{H})$ which takes the concatenated result from $\hat{\mathbf{X}}_{\mathbf{Z}}$ and $\mathbf{H}$ as input and is producing real values that are high for the given data or imputed data hat is not distinguished from the given data.

We then introduce the Lipschitz constraint, which is needed for tractability. As shown in the WGAN-GP method [30], the constraint is provided in the form of a gradient penalty. For the WGAIN-GP method we make an extension to the EM distance using the gradient penalty described in [30]. The critic C and generator G have their own weights and play an iterative two player minimax game (adversarial training). In this game the critic aims to differentiate missing values from values that where already there since the start of the training process while the generator tries to stop the critic from being able to make that separation.

In conclusion, we need to minimize the 2 loss functions below. First we look at the function we need to minimize in order to train the critic,

$$J_C = E_{\hat{\mathbf{X}}_{\mathbf{Z}},\mathbf{M},\mathbf{H}}\left[-\mathbf{M} \odot C(\hat{\mathbf{X}}_{\mathbf{Z}}, \mathbf{H})] + (1-\mathbf{M}) \odot C(\hat{\mathbf{X}}_{\mathbf{Z}}, \mathbf{H}) + \lambda_{GP}\left(\|\nabla_{\hat{\mathbf{X}}_{\mathbf{Z}}} C(\hat{\mathbf{X}}_{\mathbf{Z}}, \mathbf{H})\|_2 - 1\right)^2\right]. \quad (5)$$

Second we look at the function we need to minimize in order to train the generator,

$$J_G = E_{\hat{\mathbf{X}}_{\mathbf{Z}},\mathbf{M},\mathbf{H}}\left[-(1-\mathbf{M}) \odot C(\hat{\mathbf{X}}_{\mathbf{Z}}, \mathbf{H}) + \lambda_{MSE}\mathcal{L}_{MSE}(\hat{\mathbf{X}}_{\mathbf{Z}}, \bar{\mathbf{X}})\right], \quad (6)$$

were λ_{MSE} helps us to adapt for the impact coming from the squared loss function. Both Equation 5 and 6 update the weights w using the Adam optimization algorithm, since it improves performance for WGAN-GP according to [30].

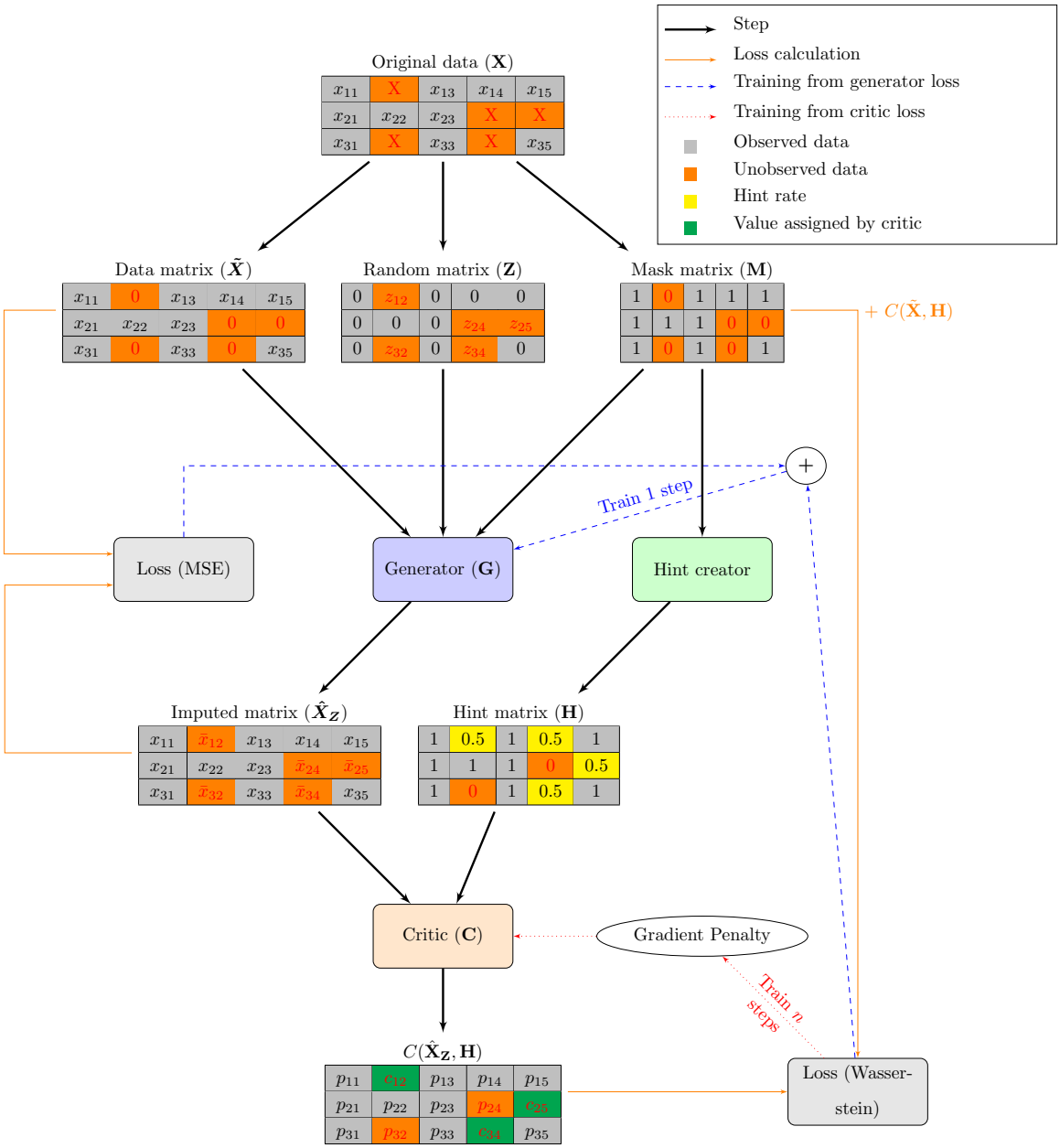


Figure 3: The overall architecture of the WGAIN-GP algorithm.

Given these loss functions, we now explain how the WGAIN-GP in its entirety can be constructed. Compared to GAIN [3], the last activation layer of the critic changes slightly. Rather than using a sigmoid activation function, a hyperbolic tangent (tanh) activation function is used. This is done, such that the same structure as in WGAIN [25] is followed. Consequently, the interpretation of the output of the critic changes. We can no longer give a probabilistic measure of how likely it is that an element is imputed or not, since the range of tanh does not have a range of 0 to 1. Rather, negative values indicate poor imputations (lower bounded by minus 1), while positive values (upper bounded by 1) indicate good imputations. This

interpretation follows the same line as in [25]. Additionally, the critic has more training epochs than the generator, as suggested in [25]. This is done to prevent the generator to dominate over the critic. Lastly, batch normalisation cannot be used whenever the Wasserstein loss with gradient penalty is used. The argumentation for this change is that the gradient penalty penalizes the norm of critic's gradient independently for every input, and not for the entire batch [30].

Algorithm 1 WGAIN-GP

Require: b, the batch size; γ, the learning rate; λ_{GP}, the gradient penalty coefficient; λ_{MSE}, the MSE coefficient

while weights have not converged **do**
  for $i = 1, ..., n_{critic}$ **do**
    Get b values from the dataset $\{\mathbf{x}_i\}_{i=1}^b$, the mask distribution $\{\mathbf{m}_i\}_{i=1}^b$ and the hint distribution $\{\mathbf{h}_i\}_{i=1}^b$
    Draw b samples from the normal distribution of $\mathbf{Z}$, $\{\mathbf{z}_i\}_{i=1}^b$
    Draw b samples from the uniform distribution $U[0,1]$, $\{\boldsymbol{\epsilon}_i\}_{i=1}^b$
    $\tilde{\mathbf{x}}_i \leftarrow \mathbf{x}_i \odot \mathbf{m}_i$;
    $\bar{\mathbf{x}}_i \leftarrow g(\tilde{\mathbf{x}}_i, \mathbf{m}_i, (1-\mathbf{m}_i) \odot \mathbf{z}_i)$;
    $\hat{\mathbf{x}}_{\mathbf{z}_i} \leftarrow \mathbf{x}_i \odot \mathbf{m}_i + \bar{\mathbf{x}}_i \odot (1-\mathbf{m}_i)$;
    Update weights of C using Adam with $\theta_1 = 0$, $\theta_2 = 0.9$, learning rate $\gamma = 10^{-4}$ and gradient:
    $\nabla J_C = \nabla[-\frac{1}{b}\sum_{i=1}^b \mathbf{m}_i \odot C(\hat{\mathbf{x}}_{\mathbf{z}_i}, \mathbf{h}_i) + \frac{1}{b}\sum_{i=1}^b (1-\mathbf{m}_i) \odot C(\hat{\mathbf{x}}_{\mathbf{z}_i}, \mathbf{h}_i)] + \frac{1}{b}\sum_{i=1}^b \lambda_{GP}(\|\nabla_{\hat{\mathbf{x}}_{\mathbf{z}_i}} C(\hat{\mathbf{x}}_{\mathbf{z}_i}, \mathbf{h}_i)\|_2 - 1)^2$;
  end for
  Get b values from the dataset $\{\mathbf{x}_i\}_{i=1}^b$, the mask distribution $\{\mathbf{m}_i\}_{i=1}^b$ and the hint distribution $\{\mathbf{h}_i\}_{i=1}^b$
  Draw b samples from the normal distribution of $\mathbf{Z}$, $\{\mathbf{z}_i\}_{i=1}^b$
  Draw b samples from the uniform distribution $U[0,1]$, $\{\boldsymbol{\epsilon}_i\}_{i=1}^b$
  $\tilde{\mathbf{x}}_i \leftarrow \mathbf{x}_i \odot \mathbf{m}_i$;
  $\bar{\mathbf{x}}_i \leftarrow g(\tilde{\mathbf{x}}_i, \mathbf{m}_i, (1-\mathbf{m}_i) \odot \mathbf{z}_i)$;
  $\hat{\mathbf{x}}_{\mathbf{z}_i} \leftarrow \mathbf{x}_i \odot \mathbf{m}_i + \bar{\mathbf{x}}_i \odot (1-\mathbf{m}_i)$;
  Update weights of G using Adam with $\theta_1 = 0$, $\theta_2 = 0.9$, learning rate $\gamma = 10^{-4}$ and gradient:
  $\nabla J_G = \nabla[-\frac{1}{b}\sum_{i=1}^b (1-\mathbf{m}_i) \odot C(\hat{\mathbf{x}}_{\mathbf{z}_i}, \mathbf{h}_i) + \lambda_{MSE}\frac{1}{b}\sum_{i=1}^b \|\bar{\mathbf{x}}_i - \hat{\mathbf{x}}_{\mathbf{z}_i}\|^2]$;
end while
return C and G

The pseudocode for this WGAIN-GP algorithm can be found in Algorithm 1. The time complexity of this algorithm is $O(b \cdot d \cdot k \cdot e)$, where b is the number of samples, d is the number of features, k is the number of layers in C and G, and e is the number of epochs. We first need to draw some samples and we need to define the hyperparameters. We choose Adam with $\theta_1 = 0$, $\theta_2 = 0.9$, we set learning rate $\gamma = 10^{-4}$, and $n_{critic} = 5$. Then, we repeat several steps until convergence of the algorithm (the weights have not changed by more than $\epsilon = 10^{-6}$). These steps include updating the weights w of C using Adam as optimization algorithm with pre-specified parameters as given in the pseudocode. Then, both the gradient of the critic and the generator need to be updated and these steps are repeated until convergence. Figure 3 gives a schematic overview of the WGAIN-GP method.

4. Data

In this section we give a description of our dataset and an overview how we prepared the data for our research. The data consists of orders from an online retailer in the Netherlands in 2019 and 2020 up until the 16th of December 2020. The term 'missing data' is used to denote data that is not yet available at that specific point in time. The same reasoning holds for data that is to be imputed: *Delivery Days*, *Case*, and *Return*. The dataset consists of both continuous and categorical data.

4.1. Data Pre-processing

The pre-processing procedure aims to prepare the data in such a way that incorrect data points are deleted, faulty data is extracted, and redundant data is removed. Furthermore, the data is transformed and new data is added. With the use of the provided date-time data, the following new variables are created: *Promised Delivery Days*, *Cancel Days*, *Shipment Days*, *Delivery Days*, *Return Days*, and *Case Days*. These variables represent the number of days between the order date and the relevant event. If an order is cancelled for example, it will not be delivered. In such a case the number -2 is used to identify this situation. In case the variable is unknown, the value -1 is used. These number classifications will help the decision tree, which will be introduced in Section 5, to identify the direction in which the order needs to go through the tree. We also set boundaries to treat non-logical data and adhere to the online retailers requirements. The promised delivery days cannot be negative and cannot exceed thirty days. Cancellations are only looked back at for a maximum of ten days after ordering, so we omit cancellations longer than this time period. The same idea holds for fulfilment, cases, and return, where the maximum is 13 days, 13 days, and 30 days when the order has been made, respectively. Based on the distribution of the total order price we only take values between €5 and €2000. By setting these boundaries we select most of the orders. Only 1% of the dataset is omitted when applying these limits. Thus, we treat orders below or above this range as outliers. Omitting those creates a smoother distribution which makes it easier for the GAIN network to learn the distribution.

The age of the seller (*Seller Age*) is added as a new feature. We take the difference between the registration date of the seller compared to the order date as age. Normalizing these ages, by minmax scaling, gives us a histogram which can be seen in Figure 4. When looking at this figure, we see that there are not that many sellers with an age above approximately 0.6, which represents the normalized value. However, since normalization depends on the maximum value, these suppliers influence the range of sellers. Therefore, we propose to take an upper boundary of 8 years as age for a seller. If a seller has an age exceeding this value, we set its age to 8 years. By doing so we decrease the maximum value, which means our normalization becomes more spread out, as can be seen when looking at Figure 5.

We now take a look at the different distributions of delivery dates (*Delivery Days*), cancellation days after the order date (*Cancel Days*), number of days before first case after order date (*Case Days*), and return days after the order data (*Return Days*). In Figure 6 we note that the distribution of delivery data is relatively spread out, but for cancellation, case, and return, we have highly imbalanced distributions.

4.1.1. Normalizing

We normalize non-categorical variables with use of a minmax scaler to make them suitable for the model. For this we normalise in such a way that all values are in a range between 0 and 1. As we explain in Section 5.4, we impute data by means of a neural network for the generator. This neural network uses a sigmoid activation function for our neural network in the final layer, which outputs values in between 0 and 1. As a result, by initially having our data in the same range, we make the conversion of renormalization much easier. The variables we normalize are the following: *Quantity Ordered*, *Total Price*, and *Seller Age*.

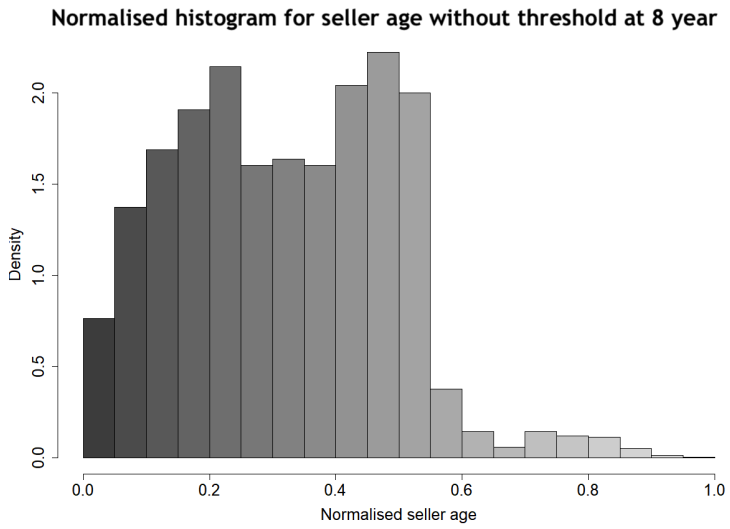


Figure 4: Normalized distribution for variable seller age if we do not take 8 years as a lower bound.

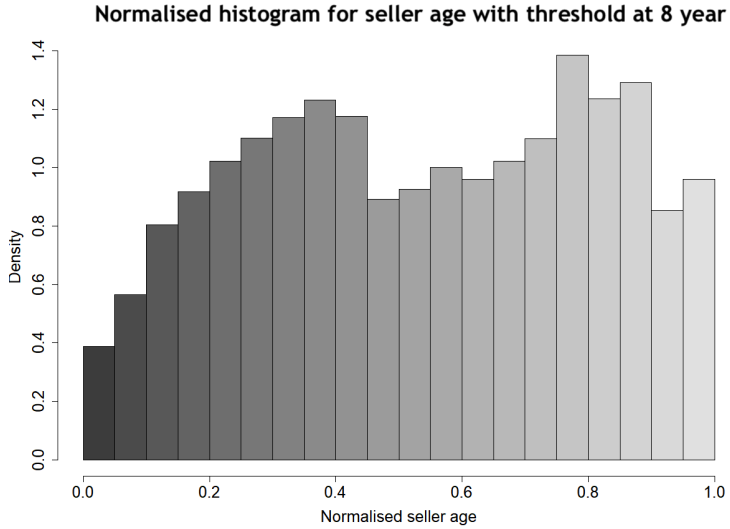


Figure 5: Normalized distribution for variable seller age if we take 8 years as a lower bound.

4.2. Adding External Variables

In order to better detect and explain patterns we find in our dataset, we add several external variables. Based on the order date we add the *Order Weekday* and the *Order Month*. This data helps us understand the time context. Furthermore, we add some holiday data as external variables. Each holiday is added as a new column, where two weeks prior to the actual holiday, the holiday value is activated by a dummy. We believe the effect of the holiday is best captured two weeks before the actual holiday. This way, four new columns are added: *Christmas, Saint Nicholas*, and *Easter*. Other external variables such as weather, temperature, and traffic jams will definitely have an impact on the delivery process. The downside is we do

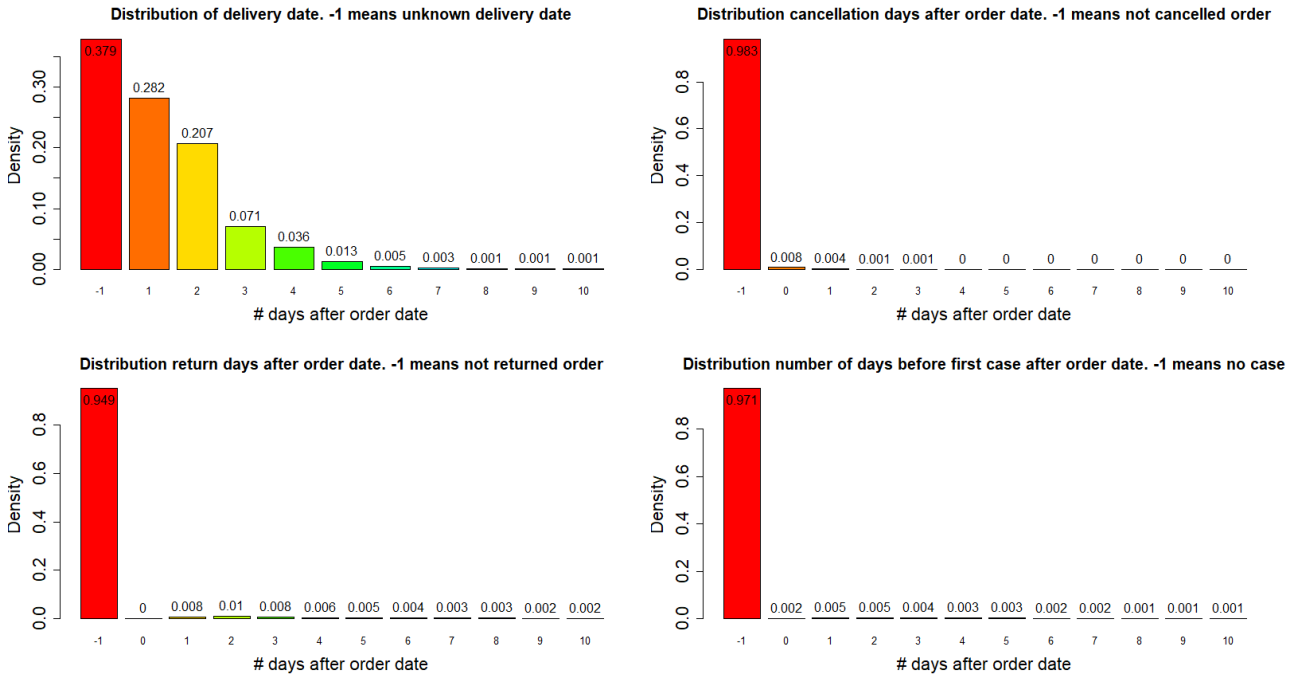


Figure 6: The distributions of delivery dates, cancellation days after the order date, number of days before first case after order date and return days after the order data.

not know the location of the supply depot and the package destination. This makes the use of these three variables not useful in our experiment, since we lack this detailed information.

4.3. Embedding Categorical Variables

In tabular data, one usually needs to make appropriate transformations for categorical variables. This implies a form of one-hot encoding, in which all levels in a categorical variable are split in different columns and subsequently are given either a value of 1 if the level matches with the observation and 0 otherwise. This approach comes with a couple of drawbacks. First of all, with a high cardinality in the levels and/or high number of observations, one would obtain sparse matrices. Secondly, one-hot encodings lead to equidistant distances between all the levels of a categorical variable, while in reality these distances may differ from level to level. As a result, using one-hot encoding makes it difficult to obtain the intrinsic properties of categorical variables. For these reasons, we propose a different type of transformation on categorical variables, named embedded layerings [31]. With embedded layerings, categorical variables are mapped into a function approximation with Euclidean distances. The mapper is trained using neural networks. In [31], it is shown that this approach yields computational advantages over traditional transformations of categorical variables. In particular, the embeddings result in better generalisations for neural networks for sparse data. Furthermore, it is shown using embeddings that result from the mappings of categorical variables into the

neural networks can be used as features in other machine learning methods. In fact, in [31] it is explained that using these embeddings as features for categorical variables could lead to higher prediction accuracy.

5. Methodology

To explain what our methodology entails, we need to explain the difference between our main methodological framework and the multiple processes that help the framework work properly. Thus, we first discuss what the main framework looks like and how we can use it to predict what label (or match) our order gets. After this, we are able to elaborate on the processes within the framework. While explaining the idea behind our framework, we consider a multi-class classification problem, a so-called matching system. In this system we aim to predict the quality of an order by labeling the order with the client being in an happy, unknown, or unhappy state. The three labels we attach to these outcomes are therefore respectively a Happy Match, an Unknown Match, and an Unhappy Match.

5.1. Framework Architecture

5.1.1. Labeling Matching Labels

In this subsection, we explain the framework architecture behind our matching system. To understand this architecture, it is important to first look at the distinctions between the different match labels we established. Therefore, we briefly summarise the characteristics and size of all these different match outcomes with the:

- **Unhappy Match (**12%**)**: All orders for which the order went wrong in at least 1 aspect. These aspects are:
    1. *Cancellation*: The order is cancelled within 10 days.
    2. *Delivery*: The order is delivered after the promised delivery date.
    3. *Return*: The order is returned within 30 days after the order takes place.
    4. *Case*: At least one case is made within 30 days after the order takes place.
- **Unknown Match (**31%**)**: All orders for which the delivery date is not known and no cancellation, return or case takes place.
- **Happy Match (**57%**)**: All orders which do not result in an Unknown or Unhappy Match.

Comparing these descriptions of the order labels, we see the classification of an order can be deduced from knowledge about the variables related to the cancellation, delivery, return, or case of an order. We call these variables the leading variables. Once these leading variables are known, it becomes clear what type of

match we have to label an order with. However, at the order date most of this information is not known. One approach to obtain information about the leading variables is waiting until information becomes available. In time, this results in a high accuracy of our predictions because we know exactly what the outcome of the matching system is. On the downside, the relevance of our predictions decreases as we need to wait a while before we can make them. In other words, there is a trade-off between the accuracy and the relevance of our prediction of the matching outcome of an order. To obtain high relevance, we need to know the outcomes of the leading variables as soon as possible, ideally right after the order is made. Therefore, we set up a framework in which we try to deduce the outcomes of these leading variables. Among other things, this framework helps us to know at what moment in time a leading variable influences the outcome of a matching label. We investigate this further with the concept of a marginal contribution for leading variables in the next section.

5.1.2. Marginal Contributions of the Leading Variables

We say a leading variable has a marginal contribution when it is able to change the label an order receives. To illustrate this, we take a random order with the label Unhappy Match. Given this label, we know the order is cancelled and/or delivered too late and/or returned and/or at least one case is made. However, we do not know which variable is actually triggering the Unhappy Match label. Leading variables triggering a match label is what we call *Active Variables*. We cannot deduce whether only one leading variable is active, or multiple. Therefore, based for the label Unhappy Match, we note the following:

The leading variable concerning the return of an order depends on the leading variable concerning the delivery date, due to the fact that a product related to the order only can be returned if it has been delivered. An order is only cancelled if it is not yet delivered. An order only can be returned whenever it is delivered. Therefore, cancellation and return cannot be active at the same time. Additionally, once an order is cancelled it cannot be delivered. As a result, cancellation and delivery cannot be active at the same time. The leading variables are able to change the label of an order to an Unhappy Match at the following moments in time:

- *Cancellation:* Between the order date and the delivery date, with a maximum of 10 days after the order date.
- *Delivery*: Once the delivery date is beyond the promised delivery date.
- *Return*: Only if the delivery date is unknown or the delivery was on time.
- *Case*: Only when the order was not cancelled, the delivery was on time and the order is not returned. Furthermore, only the first reported case changes the label.

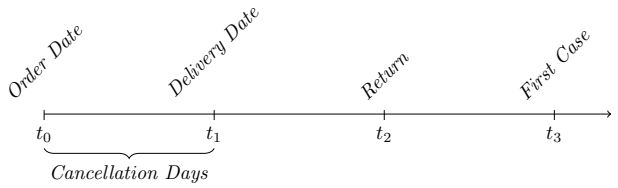


Figure 7: Timeline from the moment an order is placed at t_0 until the first case at t_3.

Based on these observations, we are able to create the following ordering of the leading variables: *Cancellation* > *Delivery* > *Return* > *Case*, where '>' means that the variable on the left occurs at an earlier moment in time than the variable on the right. We show this in a timeline in Figure 7. Now that we have these marginal contributions in place, we can create the framework architecture of our matching system.

5.1.3. Model Framework in a Tree Structure

The order of the leading variables we create based on the marginal distributions can sequentially be represented in a tree type of structure. Figure 8 shows such a tree. This tree can be decomposed into 3 different types of nodes. First of all, we have the order node. Second, we have decision nodes in which the outcome to one of the leading variables needs to be decided. Lastly, we have the outcome nodes in which the match outcome of the order is provided.

Considering Figure 8, we can see that it is possible to deduce the match label of an order using values of the leading variables. In case a leading variable becomes active for the Unhappy Match, we label the order as Unhappy. Whether an order is labeled as an Unknown Match is mostly determined by the leading variable *Delivery Date*. This variable can be decomposed in a categorical variable with one of the following 3 classes: *Late Delivery*, *Unknown Delivery*, and *On Time Delivery*. These labels depend on the type of delivery date. In the first instance, the actual delivery date is beyond the promised delivery date, while in the second instance the delivery date of an order is not known during the order process. One example in which this occurs is for orders which are delivered by mail. In these situations, the delivery date of the order is not tracked, since no signature by the customer is required whenever the order is delivered. Consequently, the order date is set to unknown. The last label corresponds to orders in which the delivery date is before or at most at the same date as the promised delivery date.

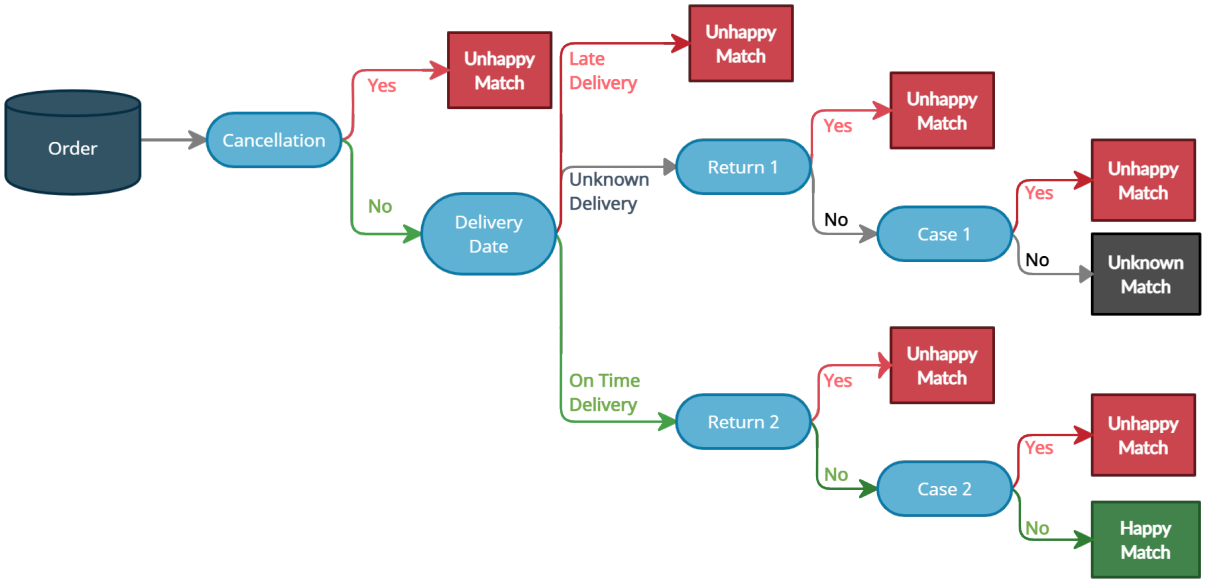


Figure 8: The model framework in a tree architecture. The numbers after Return and Case indicate the distinct return/case situation in the tree.

Once values of the leading variables are known, Figure 8 shows us different roads for classifying an order. However when these leading variables are still unknown, the tree does not provide any information about how information from these lagged variables can be obtained. In order to find this information, we need to recall the importance of the accuracy versus relevance trade-off. If we are only interested in accurate predictions, the best option is to wait until true information comes in. However, this decreases the relevance of the predictions we make. An alternative approach is to find methods able to predict outcomes of the leading variables as soon as possible, accepting the risk of making less accurate predictions. We assume our goal is to find predictions of outcomes of the leading variables immediately after an order is made such that we can make the framework in Figure 8 concrete. We name this the *Relevance Assumption*.

5.2. Concrete Model Instantiation

5.2.1. Empirical Distributions & Prediction Method Decisions

We need to decide how we want to predict outcomes of the leading variables. To do so it is important to understand how we can treat every leading variable. First of all, in the tree in Figure 8 we see that the outcomes of all leading variables can be represented as discrete choices. Additionally, we see that these discrete choices are of two types: binary decisions or decisions consisting of more than 2 levels (3).

More concretely, we convert the outcomes for the leading variables *Cancellation*, *Return* and *Case* to a binary decision outcome, while we attribute to the outcome for the leading variable *Delivery Date* three

Leading Variable	Yes	No
Cancellation	0.003	0.997
Return 1	0.063	0.937
Case 1	0.049	0.951
Return 2	0.061	0.939
Case 2	0.023	0.977

Table 2: Empirical distributions (over the years 2019 and 2020) for the leading variables as depicted in Figure 8. 'Yes' corresponds to activation of a leading variable. The numbers after Return and Case indicate the 'unknown delivery'-path (1) and 'on time delivery'-path (2).

different levels. In order to understand how likely all these outcomes are, we provide the empirical distributions of all leading variables. As such, we consider the probabilities for the leading variables with binary outcomes in Table 2 and the outcomes for the leading variables *Delivery Date* in Table 3.

Based on the results of Table 2, we see imbalanced empirical distributions for the leading variables *Cancellation*, *Return*, and *Case*. This is most apparent for the leading variable *Cancellation*, which has a probability of 0.997 of no cancellation occurring. Consequently, if we rely on binomial sampling for predicting the outcome of *Cancellation*, we expect that the probability of making a wrong prediction is close to 0. For this reason, we decide to use multinomial sampling for the prediction of this leading variable. For the empirical distributions related to the leading variable *Return*, we see again a high level of bias towards no activation, although these probabilities of not being activated are lower than for the leading variable *Cancellation*. With empirical probabilities 0.937 and 0.939 of no return occurring for *Return 1* and *Return 2* respectively, the probability of making a wrong prediction is still low but not expected to be close to 0. As a result, we apply a more advanced prediction method to find the outcome of this leading variable. When we consider the empirical distributions for the outcomes of the leading variable *Case*, we find levels of imbalance in between those of *Cancellation* and *Return*. In particular, we see that the empirical probability of no activation for *Case 1* has value 0.951, which is in the neighbourhood of the empirical distributions corresponding to *Return 1* and *Return 2*. The empirical probability of no activation for *Case 2* has value 0.977 and is closer to that of *Cancellation*. Since we expect the probability of making a wrong prediction for these two instances of the leading variable *Case* to be small but not close to 0, we use a more advanced prediction method to determine the outcome of this leading variable.

Level	Empirical Probability
Late Delivery	0.036
Unknown Delivery Date	0.338
On Time Delivery	0.626

Table 3: Empirical distributions (over the years 2019 and 2020) of the different levels for the leading variable *Delivery Date* as depicted in Figure 8.

Next, we consider the empirical distributions related to the leading variable *Delivery Date* in Table 3.

Based on these results, we note that *On Time Delivery* occurs most of the time, followed by *Unknown Delivery Date*. On the other hand, *Late Delivery* seems to have a much lower empirical probability of occurring. For the prediction of this leading variable, we also use a more advanced prediction method.

5.2.2. Model Tree using WGAIN-GP

Based on the above argumentation, we conclude that we use binomial sampling for the prediction of the outcomes of the leading variable *Cancellation*, while we use a more advanced prediction method for all other leading variables. In this paper, we make use of our newly introduced generative model WGAIN-GP in order to predict the outcomes of these leading variables. With the use of this prediction method, we obtain a concrete model instance compared to the general model framework of Figure 8. In Figure 9, we apply multinomial sampling method as the prediction method for the decision node related to the leading variable *Cancellation*. For the other decision nodes, we use WGAIN-GP. However, as the WGAIN-GP method has never been applied before, we also implement a model similar to the one in Figure 9 with the WGAIN method and the GAIN method. Adding the GAIN method seems interesting as it uses a different loss function than WGAIN and WGAIN-GP. Since GAIN, WGAIN, and WGAIN-GP have similarities, we want to have a baseline to compare these three methods. We take the most simple baseline possible, where we apply multinomial sampling to the nodes where we applied WGAIN-GP, WGAIN, or GAIN. This helps to see if our advanced methods can outperform a relatively simple model.

Next, we elaborate on the the training procedure of the GAIN, WGAIN, and WGAIN-GP methods. From now on for brevity of notation, the three methods are concatenated together under the name GAIN if we do not specify any further. In this paper, we use a combination of multinomial sampling and GAIN imputation to make predictions of the activation of the leading variables. However, the model framework of Figure 8 can also be used in conjunction with other prediction/imputation methods.

5.3. Test and Training Regime

5.3.1. Training and Test-split

We use a sample of 945,362 observations. This sample is split into two parts. One part is used for training decision nodes corresponding to predictions of the leading variables, while the second part is used to test performance of the overall tree model. Here, we use 2/3 (631,287) for the evaluation of the overall model, while we use the other 1/3 (314,075) to train prediction methods.

5.3.2. Training Data

Training Datasets. In order to impute outcomes of the leading variables *Delivery Date*, *Return*, and *Case*, it is important to take into account the relative ordering of the these leading variables. For example, the

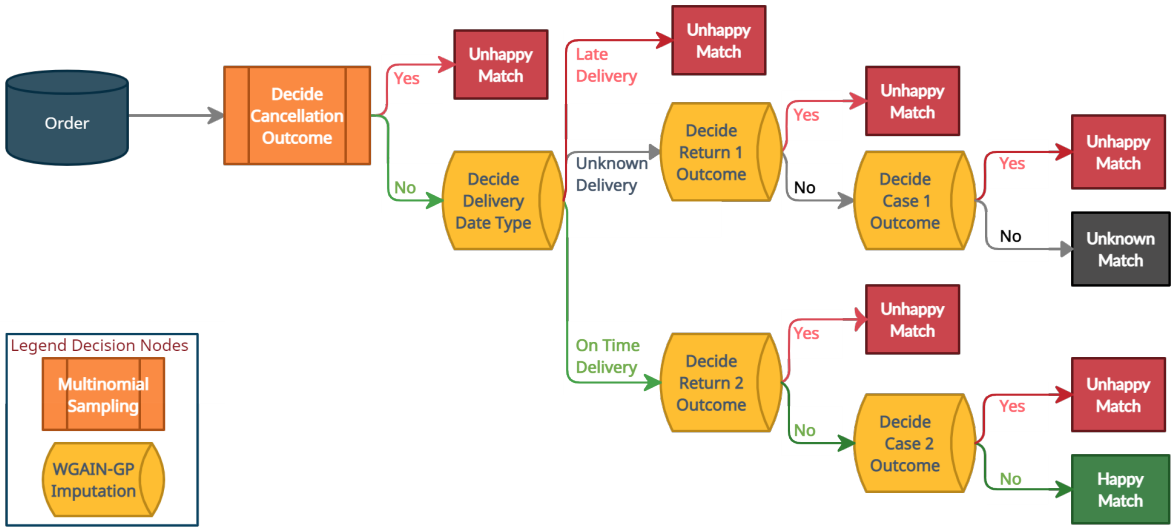


Figure 9: The decision tree used to classify an order as one of the matches: Happy Match, Unhappy Match, or Unknown Match. The percentages are probabilities used to end up in a specific node and are calculated based on the conditional probability of ending up in this node.

imputation for the leading variable *Delivery Date* depends on the outcome of the leading variable *Cancellation*. In this situation it only seems logical to impute the outcome of the leading variable *Delivery Date* if an order is not canceled. Thus, we only require the information of all non-canceled orders whenever we want to impute the outcomes for the leading variable *Delivery Date*. Similarly, for imputation of the leading variables *Delivery Date* and *Case* we need to take the prior information into account. Here, we can think of a situation in which the imputation for the leading variable *Return* relies on the outcome of the the leading variable *Delivery Date*. In case an order is delivered late, we do not need to impute the outcome for the leading variable *Return* as this does not have any marginal contribution. Whenever we want to impute the outcome of leading variable *Return*, it is therefore important to understand which information to use. The two examples above illustrate the importance of using information for all the decision nodes in Figure 9 with caution. Since it is not desired to use the same information everywhere, we only use conditioned data. The idea is to remove parts of the data which cannot be used to impute the outcomes of the corresponding leading variables. For example, when we impute the outcome for the leading variable *Delivery Date* we remove all canceled orders. Similarly, when we impute the outcome for the leading variable *Return* we only use information of all orders with the corresponding delivery date. Here, for *Return 1* we use information of all orders with an unknown delivery date, while for *Return 2* we only use information of all orders with an on time delivery date. As a result of this procedure, the number of observations used for leading variables, which rely more on other leading variables, is lower. In Table 4 we show the number of observations used for all decision nodes in Figure 9.

In Table 4, we see that from the total training sample of 314,075 observations, we use 305,808 for the training of the leading variable *Delivery Date*. Then, of these 305,808 observations, we use 102,816 observations for the training of *Return 1* and 191,851 for *Return 2*. Note that these two numbers do not add up to 314,075, since some observations are removed due to the label *Late Delivery*. Lastly, of the 102,816 observations used of *Return 1*, 96,3476 are used for the training of *Case 1*. On the other hand, of the 191,851 observations of *Return 2* 180,181 are used for *Case 2*.

Leading Variable	Number of Observations
Delivery Date	305,808
Return 1	102,816
Case 1	96,3476
Return 2	191,851
Case 2	180,181

Table 4: Data splits over decision nodes used for the imputation of the leading variables

Validation Performance on Training Data. During the imputation training process of the leading variables the data is split into 5 folds. Based on these folds we apply stratified 5-cross validation to evaluate the performance under different settings of the hyperparameters. To determine the best hyperparameters a grid search over a range of hyperparameter values is done. In order to evaluate the performance of the GAIN methods under a set of hyperparameters, we consider two types of performances. For the first type of performance, we look at a metric directly related to training data, while the second metric uses validation data.

Evaluation on the Training Set: For the training data, our goal is to obtain a generator able to generate realistic samples which can be used for data imputation. As such, we train a generator and discriminator/critic accordingly. In order to evaluate the performance of this training process, we evaluate how well the generated samples resemble the distribution of the empirical data. Secondly, we investigate if the training process has converged by means of a stopping criterion.

Distributional Resemblance:. During training of a GAIN, we need to consider the performance. To this end we can consider how well the distribution of the imputed data resembles the actual empirical distribution of the data. This can be visualised by means of a histogram. However, these histograms do not provide us with any numeric estimation. Fortunately, we can still obtain a numeric valuation, if we convert the histograms into Precision Recall Distribution plots (PRD-plots), as is proposed in [32]. First of all, we need two distributions: a reference distribution p and a learned distribution q. In our instance, p corresponds to the true empirical distribution, while q corresponds to the distribution obtained by one of our GAINs. Based on these two distributions, precision intuitively measures the quality of samples from q, while recall measures the proportion of p that is covered by q. Hence, we can say that precision measures the quality of

the generated samples, while recall measures the diversity of the generated samples. Since this is based on a distribution, we do not obtain a single point but rather an entire set of points that can be visualised into plots. PRD-plots can be converted into a numeric value, if we consider the area under the curve (AUC) of these plots. Since PRD-plots are embedded in squares with a length and width of 1, the maximum AUC equals one, while the minimum AUC equals zero. Therefore, if we convert the PRD-plots to the corresponding AUC-values, we have a numeric interpretation of the resemblance of the imputed data distribution compared to the actual data distribution.

Stopping Criterion:. A stopping criterion is needed when training the GAIN. Adding more epochs after training has already converged does not add value, so we add another convergence criterion besides the number of training epochs. This criterion is based on the loss function of the generator. The goal of the stopping criterion is to control if the loss of the generator has converged. In order to do so, we track the relative change of the loss of the generator. In case this relative change in absolute value is smaller than the small number ϵ (in which we set $\epsilon = 10^{-6}$), we might have evidence that the loss of the generator has converged. However, to make this claim only after one epoch might be too optimistic. Therefore, we add a counter which tracks how often the relative change is subsequently smaller than ϵ. In case this happens more than a specified number of times, we stop training. We set this predetermined number equal to 50 and the maximum number of training epochs equal to 1000.

Evaluation on the Validation Set: The goal of evaluation on the test set is to see how the model predicts on unseen data. We denote the True Positive as TP, the True Negative as TN, the False Positive as FP, and the False Negative as FN. We then use the following metrics:

- Accuracy $= \frac{TP+TN}{TP+TN+FP+FN}$
- Recall $= \frac{TP}{TP+FN}$
- Precision $= \frac{TP}{TP+FP}$
- $\mathrm{F}_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision}+\text{Recall}}$
- Weighted F_1: Based on the F_1-score, we can compute a weighted F_1-score by computing the F_1-score for every label in a categorical variable and subsequently multiplying them by the probability of this label occurring within the categorical variable. Say that a categorical variable has C of such labels and that π_c is the gold test data probability of label c occurring, where $c = 1, 2, ..., C$. Then the weighted F_1 score can be calculated as follows: Weighted $\mathrm{F}_1 = \sum_{c=1}^{C} \pi_c \mathrm{F}_{1c}$, where $\sum_{c=1}^{C} \pi_c = 1$.

To evaluate the training of the GAIN methods on the test set we use the weighted F_1-score, while for the evaluation of the overall model tree we use all discussed metrics.

With the use of these metrics the overall evaluation procedure is as follows. First we train one of the GAIN methods on one of the training dataset, for a prespecified number of epochs. During these epochs, the goal is

to return a generator which has the best distributional resemblance. The best distributional resemblance is given by means of the AUC value. After the generator with the best distributional resemblance is found the next step is to check the imputation quality of this generator. For this we use the validation data, in which we measure the imputation quality by means of the weighted F_1-score. The motivation for the usage of the AUC-score for the training data and the weighted F_1-score on the validation set and not the other way around is as follows. During training, we update the weights and biases of the generator multiple times. As a result, we encounter different versions of the generator, some of which have better imputation quality than others. However, better imputation quality does not necessarily imply a better distributional resemblance. In some instances, the imputation quality can be improved by only predicting one of the outcomes (low distributional resemblance). This is especially true in case one of the outcomes is overrepresented. Therefore, using the weighted F_1-score for finding the best generator might yield a generator which has relatively high imputation quality but bad distributional resemblance. In order to prevent this bias from occurring, on the training data we use the AUC-score as evaluation measure.

Both F_1-score and AUC are useful to report for imbalanced data. As our variables are imbalanced we decided to report these measures next to accuracy.

5.4. GAIN Implementation on Dataset

The GAIN framework is used when we try to predict the outcomes of the leading variables *Delivery Date*, *Return*, and *Case* at the order date. The idea is to consider the outcomes of these leading variables as missing data that we want to impute. We assume that we can impute the missing data by using the information of relevant features. In our particular situation, this corresponds to using data available at the order date provided in Table 5. In this table we first state the names of all variables we intend to use. Secondly, we mention the types of the variables. Besides continuous, count, and dummy variables, we also incorporate categorical embedding variables. The latter are embedded using embedding layers.

5.4.1. Embedding Layers

In Table 5, we have multiple categorical variables. Some of these have multiple levels (more than two), while others have exactly 2 levels (dummies). For all categorical variables with more than two levels which are used as features, we apply embedding layers.

In order to build the adapted GAIN network, we need to add embedding layers within the GAIN architecture. Usually the embedding layers are placed in front of the first layer of the model. In our situation, this implies that we need to add the embedding layer to the network corresponding to the generator. As a result, the weights of the embedding layers are trained using the loss function of the generator. Adding this embedding layer in front changes the network architecture in the following way:

Feature	Variable Type	Function
Case	Dummy	Imputation
Christmas	Dummy	Feature
Country Availability	Dummy	Feature
Country Origin	Dummy	Feature
Delivery Date	Categorical (3 levels)	Imputation
Easter	Dummy	Feature
Fulfillment Type	Dummy	Feature
Order Month	Categorical Embedding (12 levels)	Feature
Order Week	Categorical Embedding (52 levels)	Feature
Order Weekday	Categorical Embedding (7 levels)	Feature
Product Group	Categorical Embedding (60 levels)	Feature
Promised Delivery Days	Count	Feature
Quantity Ordered	Count	Feature
Saint Nicholas	Dummy	Feature
Seller Age	Count	Feature
Total Price	Continuous	Feature
Return	Dummy	Imputation
Year	Dummy	Feature

Table 5: The used variables: names, types, and functions

1. *We need to decide upfront which variables we want to embed and which ones not.* Here we make the assumption that we cannot embed variables which we want to impute, because then we would try to impute values we are training.
2. *We only add random noise to the variables we do not embed.* Since we are not imputing the variables we are embedding, we do not have to add random noise to the embedded variables.

The output dimension of the embedding layers is not necessarily fixed. Consequently, the output dimension of the embedding layers can be treated as a hyperparameter. We discuss the other hyperparameters next.

5.4.2. Hyperparameters and Hyperparameter Tuning

To train the GAIN methods we need to consider hyperparameters. Our approach in finding the correct hyperparameters is to vary some, while keeping others fixed. The main reason why we use this approach is to save computational time. In order to determine which variables we need to vary, we apply semi-supervised pre-training. During this training we try several values of a single hyperparameter using three different reasonably high batch-sizes. These batch sizes are 1024, 2048 and 5096. Using these batch sizes, we observe how a hyperparameter affects the training process. This is what we call *ceteris paribus testing*. The hyperparameters are listed below:

Batch Size: Generally smaller batch sizes lead to more stochastic behaviour and they are less expensive in terms of memory used. Larger batch sizes, on the other hand, result in less stochastic behavior and require more memory. We use the batch size as hyperparameter and also use it for *ceteris paribus testing*.

Hint Rate: The hint rate states the fraction of observations of which we provide the state to the discriminator/critic. Although we can treat the hint rate as a hyperparameter for which we can try different

values, we observed from empirical testing that changing the hint rate in a *ceteris-paribus* fashion does not have a large influence on the output results. Since the hint rate does not seem to influence the results much, we keep this hyperparameter fixed and select a hint rate of 0.5. This is based on the hint rate used in [3].

Output Dimension of the Embeddings: The output dimension of embedding layers is a hyperparameter, since it is not fixed beforehand. However, an important question to ask is whether changing output dimension significantly influences the process of training a GAIN method. In order to test this, we apply *ceteris paribus* testing. Using the smallest possible output dimension, which is equal to one, an output dimension of five, and the largest possible output dimension (the number of levels) of the categorical variable, the training process and accuracy performance do not change significantly. However, the computational time increases since the dimension of the problem increases. Therefore, in order to save computational time, we use an output dimension of one everywhere.

λ-penalties: We have two λ-penalties, one concerning the MSE loss of the generator and one concerning the gradient penalty loss of the critic. Note the latter penalty is only added for the WGAIN-GP method. The effect of a higher λ_{MSE} is putting emphasis on the MSE loss of the generator, while a higher level of λ_{GP} puts more restrictions on deviating from the Lipschitz-1 constraint. From empirical testing in a *ceteris-paribus* fashion, we observe that different values of λ_{MSE} have a similar impact on the training process. Therefore, we keep this parameter fixed and use a value of 100. This value originates from the original GAIN work [3].

Neural Network: Within the generator and discriminator/critic, general neural network hyperparameters are described. *Number of Layers (Discrete Choice):* The number of layers can be treated as a hyperparameter. We use a simple feed forward neural network with 1 hidden layer. This is based on the structure of the original GAIN [3]. *Output Dimension of Layers (Discrete Choice):* The output dimension of every layer in the neural network (except for the last layer of the discriminator/critic) can vary and therefore can be treated as a hyperparameter. However, we intend to keep these dimensions fixed and rely on the structure given in [3]. *Type of Activation Function for Layers (Discrete Choice):* Within every layer we can choose a different activation function. For example, the ReLU, tanh or sigmoid activation can be used. For different activation functions the performance of the model in terms of AUC and weighted F_1-score can change. In this paper, we fix the activation functions for all layers. In the next subsection we explain this model set-up. *The Use of Batch/Layer Normalization (Binary Choice):* By introducing batch/layer normalization to our generator and discriminator/critic, the training process of one GAIN method changes considerably. Moreover, the activation function of the last layer of the generator must be bounded between 0 and 1 for our method to work properly. This is because we have normalized our data in a range between 0 and 1 (we use a sigmoid activation for the last layer), whereas batch normalization standardizes the data. Therefore, if the last layer of the generator does not transform the output to a value between 0 and 1, the

output will go beyond the actual output when we denormalize the data. This is especially true for the ReLU activation function, which is unbounded.

5.4.3. Layers of the Neural Networks

In this paper, we use a neural network structure similar to the original GAIN paper [3]. To explain what this means we start with our data which is represented by means of a data matrix. Every column represents a feature (or variable). All these features need to pass through the GAIN methods in their entirety. For this to work well, the neural networks in the form of the generator and discriminator/critic need to know beforehand what the input dimension of the data is. This input dimension can be derived by considering the number of features in our data matrix. Given this number, we multiply it by two to obtain the correct input dimension. The reason why we need to multiply it by two is that the mask matrix of our data matrix needs to pass through the GAIN methods as well. Therefore, we concatenate the data matrix and mask matrix and pass them through the GAIN methods. From now on we abbreviate this input dimension to Dim. Consequently, we make use of a feedforward neural network with one hidden layer for both the generator and the discriminator/critic. In [3] dense layers are used everywhere, with a hyperbolic tangent (tanh) activation function for all layers besides the last one. In this paper we deviate from this by replacing the tanh activation functions with ReLU activation functions. We prefer ReLU over tanh to prevent saturation from occurring. Furthermore, when a critic is involved in our imputation method we apply a tanh function for the last layer as proposed in [25].

As a last deviation from the original paper, we apply batch normalisation for the generator and discriminator and layer normalisation for the critic. The argumentation for the inclusion of these two types of normalization is as follows. Although the data which enters a neural network usually is normalized beforehand, this does not necessarily imply that the normalized range is attained when the data is transformed within the layers. The reason for this is that certain activation functions can stretch out the original input data to go far beyond the normalized range. Here we can for example think about the ReLU activation function, which is unbounded. As a result, when data passes from one layer to the other it is possible that

Generator	Type of Layer	Activation Function	Output Dimension
L_1	Dense Layer & Batch Normalization	ReLU	Dim
L_2	Dense Layer & Batch Normalization	ReLU	$\lfloor \frac{\text{Dim}}{2} \rfloor$
L_3	Dense Layer	Sigmoid	Dim
Discriminator	Type of Layer	Activation Function	Output Dimension
L_1	Dense Layer & Batch Normalization	ReLU	Dim
L_2	Dense Layer & Batch Normalization	ReLU	$\lfloor \frac{\text{Dim}}{2} \rfloor$
L_3	Dense Layer	Sigmoid	Dim
Critic	Type of Layer	Activation Function	Output Dimension
L_1	Dense Layer & Layer Normalization	ReLU	Dim
L_2	Dense Layer & Layer Normalization	ReLU	$\lfloor \frac{\text{Dim}}{2} \rfloor$
L_3	Dense Layer	Tanh	Dim

Table 6: Network layers for the Generator, Discriminator (GAIN) and Critic (WGAIN(-GP)). L_i, denotes the layer for $i = 1, 2, 3$. Dim corresponds to the number of features in our data matrix.

the data is not normalized anymore. To solve this issue, batch normalization can be added to a neural network. With batch normalization, we add a batch normalization layer, which is placed after a layer with an activation function. In this layer, the batch is normalized by subtracting the mean and dividing by the standard error, such that the normalized mean is 0 and the normalised standard deviation 1 [33].

As an alternative to batch normalization, layer normalization also can be applied. With layer normalization, we change the batch normalization layer to a layer normalization layer. The difference between the layer normalization layer with the batch normalisation layer is that this layer does not normalize the batches. Rather, the normalization is applied on all the features. If we represent our input data as a matrix in which the rows correspond to observations and the columns to features, we can therefore say that batch normalization normalizes with respect to the rows, whereas layer normalization normalizes with respect to the columns [34]. Table 6 shows networks for respectively the generator, discriminator, and critic. Note that the output dimensions of the layers are similar to the ones used by [3] and [13].

6. Results

In this section we present the results of our methodology. The results are split into four parts. In the first part, we discuss the results related to the imputation of the leading variables separately, while in the second part we elaborate upon the results of the overall model tree. In the third part, the results for the variables related to the leading ones. Last, in the fourth part, we give suggestions on how to further improve the results.

6.1. Imputation Quality of the Leading Variables

In Table 7 we present the results regarding the imputation quality of the leading variables. For every leading variable, the results in the first column correspond to the baseline. Whenever we use the baseline, we apply multinomial sampling. As we base the probabilities of multinomial sampling on the empirical probabilities of Tables 2 and 3, the AUC-scores are equal to 1 for these.

Before we explain the results of Table 7, two remarks need to be made. First of all, for the imputation of the leading variable *Cancellation*, we use multinomial sampling under all methods, which is why all results are the same. Secondly, since all methods have a level of stochasticity within them, we rely on the usage of a seed in order to replicate results. The seed value is set to 5 everywhere.

Cancellation. For the imputation regarding the leading variable *Cancellation*, we see high accuracies and weighted F_1-scores. This is not surprising, given that 99.7% of all orders are not canceled. Also note that the AUC-score is 1 everywhere, since the empirical distribution is used to generate the outcomes.

	Cancellation				*Delivery Date*				*Return 1*			
	Base	GAIN	WGAIN	W-GP	Base	GAIN	WGAIN	W-GP	Base	GAIN	WGAIN	W-GP
Batch-Size	*	*	*	*	*	2048	2048	1024	*	2048	2048	2048
AUC	1	1	1	1	1	0.992	**0.995**	*0.974*	1	*0.998*	**1**	**1**
Accuracy	**0.991**	**0.991**	**0.991**	**0.991**	*0.428*	0.641	0.610	**0.709**	0.869	*0.855*	0.866	**0.874**
Weighted F_1	**0.991**	**0.991**	**0.991**	**0.991**	*0.429*	0.581	0.543	**0.664**	0.875	*0.868*	0.874	**0.879**
	Case 1				*Return 2*				*Case 2*			
	Base	GAIN	WGAIN	W-GP	Base	GAIN	WGAIN	W-GP	Base	GAIN	WGAIN	W-GP
Batch-Size	*	2048	2048	2048	*	2048	2048	2048	*	2048	2048	2048
AUC	1	1	1	0.998	1	1	1	0.981	1	1	1	1
Accuracy	**0.920**	*0.824*	0.919	0.914	0.870	**0.891**	*0.866*	0.872	0.944	*0.935*	0.940	**0.951**
Weighted F_1	0.913	*0.861*	**0.914**	0.912	0.875	*0.868*	0.874	**0.879**	0.913	*0.861*	**0.914**	0.912

Table 7: Results of GAIN of imputation at the order date. The symbol * indicates irrelevance of the batch-size (multinomial sampling). Text in **bold** indicates best results while results in *italics* indicate worst results. Base = baseline; W-GP = WGAIN-GP.

Delivery Date. When we consider the results of the leading variable *Delivery Date*, we first of all see AUC-scores close to 1 for all GAIN methods. This indicates that all GAIN methods resemble the underlying empirical distribution closely. In terms of accuracy and weigthed F_1-scores, we see that all GAIN methods outperform the baseline. In terms of accuracy all GAIN methods improve at least upon the baseline with 18.2 percentage points, while for the weighted F_1-score the GAIN methods outperform the baseline with at least 11.4 percentage points. Additionally, WGAIN-GP outperforms both GAIN and WGAIN in terms of these two measures. The accuracy of WGAIN-GP is 6.8 percentage points higher than that of the second best GAIN method, while the weighted F_1-score of WGAIN-GP exceeds the second best GAIN method by 8.3 percentage points.

Return. In order to evaluate the results of the leading variable *Return*, we need to consider the results for both *Return 1* and *Return 2* as depicted in Table 7. For this, let us first elaborate upon the results for *Return 1*. To start off, we see that the AUC-scores of all GAIN methods are close to 1, or rounded to three decimals equal to 1. Comparing these results to those of the leading variable *Delivery Date*, we see that these AUC-scores are higher in the case for *Return 1*. In order to understand why this is the case, we need to recall the binary nature of the leading variable *Return*. Since this leading variable consists of two levels, it is easier to obtain higher AUC compared to a (leading) variable with more than two levels (*Delivery Date*). In terms of accuracy and weighted F_1-score, we have that all GAIN methods have performance which are in line with the baseline. Inspecting these results more closely, we see that WGAIN-GP outperforms all methods marginally, while the original GAIN performs worst. By moving to the results of *Return 2*, we see results which are comparable to those of the results of *Return 1*. Similarly to the results of *Return 1*, we find AUC-scores close or equal to 1 and accuracy and weighted F_1-scores comparable with the baseline. Inspecting these results more closely, we find that the accuracy of GAIN is highest, while the weighted F_1-score for WGAIN-GP is higher than all other methods.

Case. Similar to the results the leading variable *Return*, in order to evaluate the results of the leading variable *Case* we need to consider the results for both *Case 1* and *Case 2*. For this purpose, we first examine the results of *Case 1*. In terms of AUC-scores, we see all GAIN scores have AUC-scores close to 1 or rounded equal to 1. The argumentation for these values is the same as for the leading variable *Return*: the leading variable *Case* has two levels. When we move to imputation quality measures by means of the accuracy and weighted F_1-scores, we first of all note that the original GAIN method performs worse than the baseline. For the other two GAIN methods, we have similar performances compared to the baseline. If we consider the results of *Case 2*, we are able to draw conclusions analogous to those of *Case 1*. First of all, the rounded AUC-scores are all equal to 1. Moreover, the imputation quality of GAIN is worse than the baseline, whereas the results of WGAIN and WGAN-GP are more in line with the baseline. More precisely, we find an accuracy of WGAIN-GP which exceeds the baseline by 0.007 percentage points.

6.2. Results on Main Model

6.2.1. Distributional Resemblance

In Table 8, we show the distributions over all match labels using all different imputation methods. In brackets we show the deviations per match label from the actual distributions. In the last column we provide the mean absolute deviation (MAD) compared to the actual distribution. Based on the results of Table 8, let us provide an overview of all results.

Method	Happy Match	Unknown Match	Unhappy Match	MAD
Actual	0.571 (*)	0.308 (*)	0.121 (*)	*
Baseline	0.572 (0.001)	0.299 (-0.009)	0.129 (0.008)	0.006
GAIN	0.605 (0.034)	0.286 (-0.022)	0.109 (-0.012)	0.023
WGAIN	0.617 (0.046)	0.262 (-0.046)	0.121 (0.000)	0.031
WGAIN-GP	0.557 (-0.014)	0.364 (0.056)	0.079 (-0.042)	**0.037**

Table 8: Distribution over all the match outcomes for all the different methods. The results in brackets indicate the deviation of the match label under the prediction method from the actual distribution (the maximum value is shown in bold).

Baseline. If we inspect the results of the baseline in Table 8, we see the baseline closely matches the actual distribution. The level with the largest absolute deviation is that of the Unknown Match with a value of 0.009. Furthermore the MAD equals 0.006, which indicates small deviations from the actual distribution as well.

GAIN. By examining the results of GAIN in Table 8 we observe larger deviations than under the baseline. The minimum absolute deviation of the GAIN method comes from the Unhappy Match with a value of 0.012. This value exceeds the maximum absolute deviation of the baseline (0.009). The maximum absolute

			Happy Match			Unknown Match			Unhappy Match		
	A	W-F_1	F_1	P	R	F_1	P	R	F_1	P	R
Baseline	*0.434*	*0.424*	*0.571*	*0.571*	*0.571*	*0.304*	*0.308*	*0.299*	0.123	0.120	0.128
GAIN	0.543	0.538	0.662	0.643	0.682	0.473	0.492	0.456	*0.113*	*0.119*	0.108
WGAIN	0.531	0.517	0.656	0.632	0.682	0.441	0.480	0.408	**0.131**	0.130	**0.131**
WGAIN-GP	**0.619**	**0.609**	**0.716**	**0.725**	**0.707**	**0.605**	**0.559**	**0.659**	0.115	**0.144**	*0.095*

Table 9: Results with imputation at the order date. Text in **bold** indicates best result, text in *italics* worst result.
A = Accuracy; W-F_1 = Weighted F_1; F_1 = F_1-score; P = Precision; R = Recall

deviation is equal to 0.034 and is derived from the Happy Match label. With a MAD of 0.023, we additionally see a higher MAD than under the baseline. Inspecting Table 8 more closely, we find that the Happy Match label is overrepresented whereas the Unknown and Unhappy Match labels are underrepresented.

WGAIN. Considering the results of WGAIN in Tables 8, we are able to make a couple of observations. First of all, we see the Unhappy Match label closely resembles the actual distribution since the deviation from the actual empirical distribution equals 0 if we round the distributions up to 3 decimals. In contrast, absolute deviations for the Happy and Unknown Match labels are larger with a value of 0.046 (positive for the Happy Match and negative for the Unhappy Match) for both match labels. Consequently, the MAD for WGAIN exceeds that of both the baseline and GAIN.

WGAIN-GP. In order to evaluate the distributional resemblance for WGAIN-GP compared to the actual empirical distribution, we need to have a look at the last rows of Tables 8. Based on these numbers, we see WGAIN-GP has the highest absolute deviations of all methods. The minimum absolute deviation corresponds to the Happy Match with a value of 0.014. This value is larger than the minimum absolute devations of all other prediction methods. Moreover, the largest absolute deviation coming from the Unknown Match (0.056) is larger than that of all other prediction methods. As a result the MAD of WGAIN-GP also is the largest of all prediction methods. In terms of the deviation values in Table 8, we find that the Unknown Match label is overrepresented. As a result, the labels of the Happy and Unhappy Match are underrepresented.

6.2.2. Accuracy Results

In Table 9, we present the match label results for all the imputation methods. In order to obtain an understanding for all the represented numbers, we first evaluate the overall performance. Secondly, we elaborate upon the results of the different match labels.

Overall Performance. If we have a look at the accuracies (A) and weighted F_1-scores (W-F_1) of Table 9, we are able to compare the results between all methods. Based on the results of both accuracy and weighted F_1-scores, we see all GAIN methods outperform the baseline. In addition, WGAIN-GP performs best both in terms of accuracy and weighted F_1-score with values of 0.619 and 0.609, respectively. The accuracy of

WGAIN-GP is 7.6 percentage points higher than that of the second most accurate method (GAIN). In terms of weighted F_1-scores WGAIN-GP outperforms the second best method based on this metric (GAIN) by 7.1 percentage points. In order to evaluate why WGAIN-GP performs best, it is useful to examine the results of the different match labels.

Happy Match. For the Happy Match label, we see all methods outperform the baseline in terms of F_1-score, precision and recall. Furthermore, as WGAIN-GP outperforms GAIN with a percentage point differences of 5.4 and WGAIN with a percentage points difference of 6 in terms of F_1-score, WGAIN-GP outperforms the other two GAIN methods based on this measure. In order to explain why, it is useful to take the results of precision and recall into account. For precision, we have that WGAIN-GP outperforms GAIN and WGAIN respectively by 8.2 and 9.3 percentage points, while WGAIN-GP outperforms these two methods in terms of recall by 2.5 percentage points. Thus WGAIN-GP outperforms the other two methods based on F_1-scores which can be both explained from precision and recall. Of these 2 metrics, precision seems to have the most influence, since these deviations are higher. Since the Happy Match label accounts for 57.1% of all orders, this partially explains why WGAIN-GP outperforms the other two GAIN methods in terms of accuracy and weighted F_1-score.

Unknown Match. For the Unknown Match label, we observe that all GAIN methods outperform the baseline in terms of F_1-score, precision and recall. Furthermore, WGAIN-GP outperforms GAIN and WGAIN in terms of F_1 scores, since the F_1 score of WGAIN-GP is 13.2 percentage points higher compared to GAIN and 16.4 percentage points compared to WGAIN. In order to understand why, we can have a look at the results of precision and recall under this match label. Here we find that WGAIN-GP outperforms GAIN and WGAIN in terms of precision by respectively 6.7 and 7.9 percentage points. For the measure recall, we obtain percentage point differences of 20.3 and 25.1 in favor of WGAIN-GP respectively to GAIN and WGAIN. Therefore, WGAIN-GP outperforms GAIN and WGAIN both in terms of precision and recall. Of these two metrics, recall has the largest influence on the deviation in terms of the F_1 score, as percentage point differences based on this metric are higher. As 30.8% of all orders are classified with the Unknown Match label, this gives an additional explanation for why WGAIN-GP outperforms GAIN and WGAIN in terms of accuracy and weighted F_1-score.

Unhappy Match. For the Unhappy Match label only WGAIN outperforms the baseline. Additionally, GAIN performs worst in terms of all prediction methods in terms of F_1-score. For the metric precision, we find that WGAIN-GP performs best in terms of the Unhappy Match label, while the original GAIN method performs here worst too. For recall, we find that WGAIN performs best, while WGAIN-GP performs worst.

Based on these numbers, we cannot state that the prediction of the Unhappy Match label contributes to the overall outperformance of WGAIN-GP to the other prediction methods. However, as 12.1% of all orders are classified with an Unhappy Match label, this label is underrepresented compared to the other 2 labels. Consequently, the overall influence of this match label on the overall accuracy and weighted F_1-score is lower compared to that of the Happy and Unknown Match.

6.3. Results for the Other Variables

6.3.1. Imputation Quality

In Table 10, the results of the training process of the different GAIN methods for the variables related to the leading ones are depicted. Note, the optimal configuration can be different for every variable and for every GAIN method. In this table, the AUC-values and RMSPE/weighted F_1-scores are shown.

	Shipment Days			*Transporter Type*			*Transporter Days*		
	GAIN	WGAIN	W-GP	GAIN	WGAIN	W-GP	GAIN	WGAIN	W-GP
Batch Size	1024	512	1024	512	1024	512	1024	256	256
AUC	*0.975*	0.977	**0.997**	**1**	**1**	**1**	*0.621*	**0.731**	0.642
RMSPE	0.967	**0.964**	*1.046*	$*$	$*$	$*$	*2.04*	1.778	**1.231**
Weighted F_1	$*$	$*$	$*$	*0.574*	0.646	**0.709**	$*$	$*$	$*$

Table 10: Results of GAIN of imputation at the order date using batch/layer normalization and sigmoid activation function for the last layer of the generator. $*$ indicates that this value is irrelevant. W-GP = WGAIN-GP. Text in **bold** indicates best results while results in *italics* indicate worst results.

Shipment Days. Considering the results of the variable *Shipment Days* in Table 10, we first see good performances in terms of AUC as all methods are close to 1. However, in terms of imputation quality improvements can still be made. All methods approximately have a RMSPE equal to 1. Given the range of *Shipment Days* (8), this can be considered a large value.

If we compare the methods to each other, we see WGAIN-GP has the highest AUC, while the GAIN method has the lowest RMSPE. However, all results are quite close to each other.

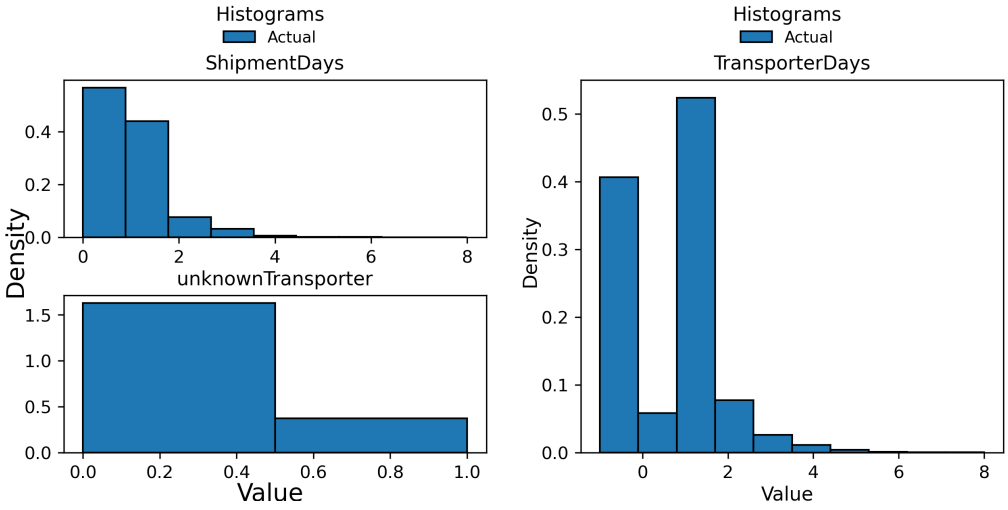
Transporter Type. If we consider the results of *Transporter Type* in Table 10, the first notable observation is the AUC score. With a rounded value (to 3 decimals) of 1, this is a perfect score. Secondly, the average weighted F_1-score greatly differs per method. GAIN performs worst, while WGAIN-GP performs best. Consequently, the quality of imputations for this variable seems to be best in the WGAIN-GP method.

Transporter Days. For the variable *Transporter Days*, we see a worse performances of both AUC and RMSPE compared to predictions made for the other two variables.

6.3.2. Limitations

Based on results of Table 10, we now elaborate on limitations of the current set-up. These can be decomposed into two separate parts. The first part has to do with the way we currently represent distributions of variables we try to impute. The second part deals with distributional resemblance compared to quality of the imputations.

Model Representation. The results of Table 10 show us a clear difference in performance of predicting the variables *Shipment Days* and *Transporter Type* compared to the variable *Transporter Days*. This holds for all methods. To understand this, we need to compare distributions related to these 2 variables with the help of Figure 10.



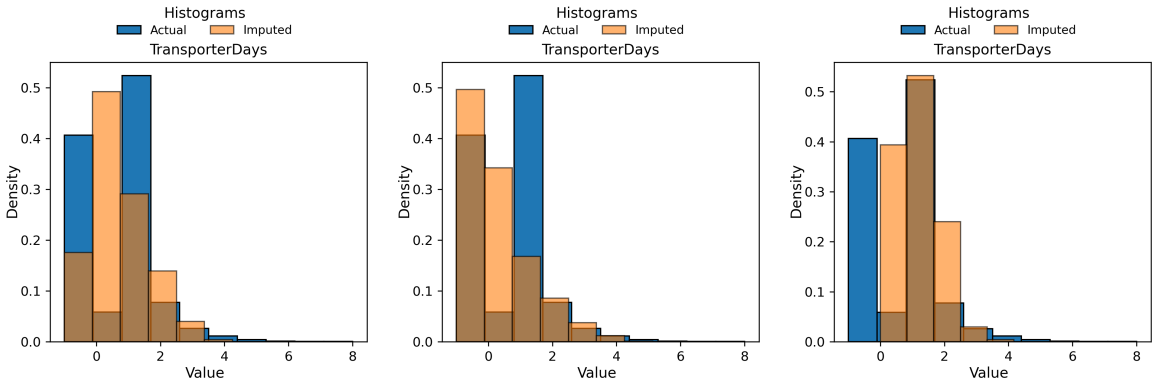
(a) (b)

Figure 10: Histograms of actual values for the variables *Shipment Days*, *Transporter Type* and *Transporter Days*.

Considering the two distributions of *Shipment Days* and *Transporter Days* in Figure 10, we see the true empirical distribution of *Shipment Days* starts with the mode and decreases with every new value. The true empirical distribution for *Transporter Days* on the other hand, starts with a mode and then suddenly drops. After this, a second mode follows and we again see a sharp drop. Finally, the values start to decrease more gradually. Therefore, comparing the two distributions we see the distribution concerning the variable *Shipment Days* can be considered easier to mimic in two aspects. Firstly, it moves more gradually, and secondly it only has 1 mode. Hence, given the shape of the two empirical distributions it seems the distribution for the variable *Shipment Days* is easier to obtain than the distribution for *Transporter Days*.

Based on these distributions and results in Table 10, we see the different GAIN methods have an easier task in finding the right distributions for the variables *Shipment Days* and *Transporter Type*. Furthermore,

since all AUC scores are nearly equal to 1 the overall shape is also expected to represent the distributions as shown in Plot (a) of Figure 10. For *Transporter Days* on the other hand, we see AUC scores are low. Therefore, we also show distributions coming from the different GAIN methods by means of Figure 11.



(a) (b) (c)

Figure 11: Histograms of imputed values under GAIN (a), WGAIN (b) and WGAIN-GP (c).

In Figure 11, we see all GAIN methods have different distributions for the variable *Transporter Days*. Therefore, it is likely the performances deviate as well.

AUC Compared to Quality of Imputations. From results in Table 10, we generally see a clear distinction between how well the distribution of imputed data resembles the empirical distribution of actual data and the quality of these imputations. We derive this from high AUC scores for the variables *Shipment Days* and *Transporter Type*, but the relatively poor values of the RMSPE and the weighted F_1-score. Therefore, the question is how we can improve the values of the RMSPE and weighted F_1-score while ideally still obtaining high AUC values. To this extend we can for example think about changing the activation functions of the generator and/or discriminator/critic.

6.4. Improvements Over Current Set-Up

Based on the explained results, we propose the following improvements for future work over our current model set-up. The dataset needs to be balanced. One of the current issues with our data is the level of high imbalance for the Unhappy Match label. In order to improve upon this, synthetic data can be added to our dataset, such that the dataset becomes more balanced. Also, the neural network structures for the generator and discriminator/critic as depicted in Table 6 might not be the best for our problem at hand. Although our methods provide high AUC values for the leading variables, the imputation quality can still be improved. This is especially true for the leading variable *Delivery Date*, which plays a key-role in determining the match labels. By improving the performance of the imputation quality of the leading variable, we can increase the

overall performance of the main model. For a lesser extent this is true for the imputation of *Return* and *Case*. Currently, we perform similarly to the baseline, so the question is if other neural network architectures can improve on the baseline. However, we see that the performances are already quite high, since the levels of these leading variables are highly imbalanced. Therefore, the most important goal here is to find a neural network architecture which improves the imputation quality for the leading variable *Delivery Date*.

7. Conclusions

This research shows that we have accurately predicted the outcomes of leading variables immediately after the order date with the use of WGAIN-GP. Using the Wasserstein distance, compared to the Jensen-Shannon divergence, leads to a better performance in training a generative adversarial imputation network. By implementing the Gradient Penalty as an addition to the WGAIN method the chance of mode collapse has drastically reduced. The GAIN methods used in this report show that the distributional resemblance, by means of AUC scores, show a similar performance. Each method is able to closely resemble the actual distribution. Measured with F_1-score metrics, the WGAIN-GP method has the highest imputation quality of all methods used. Especially the leading variable *Delivery Date* is best predicted by the WGAIN-GP method. For leading variables *Return* and *Case* we see a similar performances under all three GAIN imputation methods. In order to answer the main research question of this paper, we need to consult the results of the main model tree. Based on these results, we see WGAIN-GP is the best method in predicting the outcomes of leading variables immediately after the order date with an accuracy of 0.61 and weighted F_1-score of 0.609. Moreover, we see that all GAIN imputation methods improve upon accuracy and weighted F_1-score compared to the baseline, which is based on multinomial sampling. However, the GAIN and WGAIN methods show mediocre results as they have accuracies of 0.543 and 0.531, and weighted F_1-scores of 0.538 and 0.517, respectively.

In general, our work shows that the proposed WGAIN-GP is an improvement on the existing WGAIN in imputing tabular data. Thus, WGAIN-GP sets a new state-of-the-art in the realm of GAN-based imputation models. For our specific domain of improving customer satisfaction this is highly relevant as it allows an e-commerce company to immediately react after an order is placed to prevent customer dissatisfaction and possibly avoid customer churn.

In order to improve the current results, we can start with finding ways to make the current dataset more balanced or with using a different neural network architecture. However, the methods for the leading variables *Return* and *Case* do not outperform the baseline. This is not surprising, since these binary variables are highly imbalanced given the size of the dataset. As future work we would like to use data augmentation

methods to balance these variables [35]. On top of this, we can use a different loss function for the generator and discriminator/critic combining the loss functions for GAIN and WGAIN(-GP). For example, we can follow the suggestion in [36] which combines the Jensen-Shannon Divergence of the GAIN method with the Wasserstein Loss of the WGAIN(-GP) method.

We also would like to further investigate a more recent trend of using diffusion models for data imputation [37, 38, 39]. While GAN-based models can suffer from mode collapse and instability, diffusion models are more stable because of a well-defined likelihood objective and they do not need a critic. On the other hand, diffusion models have a slower inference time due to many forward passes, while GAN-based methods produce outputs faster as they require a single forward pass.

References

[1] B. Von Helversen, K. Abramczuk, W. Kopeć, R. Nielek, Influence of Consumer Reviews on Online Purchasing Decisions in Older and Younger Adults, Decision Support Systems 113 (2018) 1–10.

[2] G. Seni, J. Elder, Ensemble Methods in Data Mining: Improving Accuracy through Combining Predictions, Morgan & Claypool Publishers, 2010.

[3] J. Yoon, J. Jordon, M. Schaar, GAIN: Missing Data Imputation Using Generative Adversarial Nets, in: Proceedings of the 35th International Conference on Machine Learning (ICML 2018), PMLR, 2018, pp. 5689–5698.

[4] T. Emmanuel, T. Maupong, D. Mpoeleng, T. Semong, B. Mphago, O. Tabona, A Survey on Missing Data in Machine Learning, Journal of Big Data 8 (1) (2021) 140.

[5] M. Alabadla, F. Sidi, I. Ishak, H. Ibrahim, L. S. Affendey, Z. C. Ani, M. A. Jabar, U. A. Bukar, N. K. Devaraj, A. S. Muda, et al., Systematic Review of Using Machine Learning in Imputing Missing Values, IEEE Access 10 (2022) 44483–44502.

[6] L. O. Joel, W. Doorsamy, B. S. Paul, A Review of Missing Data Handling Techniques for Machine Learning, International Journal of Innovative Technology and Interdisciplinary Sciences 5 (3) (2022) 971–1005.

[7] M. Barrabés, M. Perera, V. N. Moriano, X. Giró-I-Nieto, D. M. Montserrat, A. G. Ioannidis, Advances in biomedical missing data imputation: A survey, IEEE Access 13 (2024) 16918–16932.

[8] J. L. Schafer, Analysis of Incomplete Multivariate Data, CRC Press, 1997.

[9] S. Van Buuren, Flexible Imputation of Missing Data, CRC Press, 2018.

[10] P. Jonsson, C. Wohlin, An Evaluation of k-Nearest Neighbour Imputation using Likert Data, in: Proceedings of the 10th IEEE International Software Metrics Symposium (METRICS 2004), IEEE, 2004, pp. 108–118.

[11] D. P. Kingma, M. Welling, Auto-Encoding Variational Bayes, in: Proceedings of the 2nd International Conference on Learning Representations (ICLR 2014), OpenReview.net, 2014.

[12] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative Adversarial Networks, in: Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS 2014), Curran Associates, Inc., 2014, pp. 2672–2680.

[13] M. Friedjungová, D. Vašata, M. Balatsko, M. Jiřina, Missing Features Reconstruction Using a Wasserstein Generative Adversarial Imputation Network, in: Proceedings of the 20th International Conference on Computational Science (ICCS 2020), Springer, 2020, pp. 225–239.

[14] X. Miao, Y. Wu, L. Chen, Y. Gao, J. Yin, An Experimental Survey of Missing Data Imputation Algorithms, IEEE Transactions on Knowledge and Data Engineering 35 (7) (2022) 6630–6650.

[15] Y. Sun, J. Li, Y. Xu, T. Zhang, X. Wang, Deep Learning versus Conventional Methods for Missing Data Imputation: A Review and Comparative Study, Expert Systems with Applications 227 (2023) 120201.

[16] I. Lizarralde, C. Mateos, A. Zunino, T. A. Majchrzak, T. Grønli, Discovering Web Services in Social Web Service Repositories Using Deep Variational Autoencoders, Information Processing & Management 57 (4) (2020) 102231.

[17] A. Nazabal, P. M. Olmos, Z. Ghahramani, I. Valera, Handling Incomplete Heterogeneous Data Using VAEs, Pattern Recognition 107 (2020) 107501.

[18] J. Han, Z. Zhang, N. Cummins, B. Schuller, Adversarial Training in Affective Computing and Sentiment Analysis: Recent Advances and Perspectives, IEEE Computational Intelligence Magazine 14 (2) (2019) 68–81.

[19] Y. Luo, X. Cai, Y. Zhang, J. Xu, X. Yuan, Multivariate Time Series Imputation with Generative Adversarial Networks, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS 2018), Curran Associates, Inc., 2018, pp. 1603–1614.

[20] D. Adhikari, W. Jiang, J. Zhan, Z. He, D. B. Rawat, U. Aickelin, H. A. Khorshidi, A Comprehensive Survey on Imputation of Missing Data in Internet of Things, ACM Computing Surveys 55 (7) (2022) 1–38.

[21] Y. Zhang, R. Zhang, B. Zhao, A Systematic Review of Generative Adversarial Imputation Network in Missing Data Imputation, Neural Computing and Applications 35 (27) (2023) 19685–19705.

[22] W. Dong, D. Y. T. Fong, J.-s. Yoon, E. Y. F. Wan, L. E. Bedford, E. H. M. Tang, C. L. K. Lam, Generative Adversarial Networks for Imputing Missing Data for Big Data Clinical Research, BMC Medical Research Methodology 21 (2021) 1–10.

[23] R. Shahbazian, S. Greco, Generative Adversarial Networks Assist Missing Data Imputation: A Comprehensive Survey & Evaluation, IEEE Access 11 (2023) 88908–8828.

[24] J. Kim, D. Tae, J. Seok, A Survey of Missing Data Imputation Using Generative Adversarial Networks, in: Proceedings of the 2020 International conference on artificial intelligence in information and communication (ICAIIC 2020), IEEE, 2020, pp. 454–456.

[25] M. Arjovsky, S. Chintala, L. Bottou, Wasserstein Generative Adversarial Networks, in: Proceedings of the 34th International Conference on Machine Learning (ICML 2017), PMLR, 2017, pp. 214–223.

[26] S. C.-X. Li, B. Jiang, B. Marlin, MisGAN: Learning from Incomplete Data with Generative Adversarial Networks (2019).

[27] C. Shang, A. Palmer, J. Sun, K.-S. Chen, J. Lu, J. Bi, VIGAN: Missing View Imputation with Generative Adversarial Networks, in: Proceedings of the 4th IEEE International Conference on Big Data (IEEE Big Data 2017), IEEE, 2017, pp. 766–775.

[28] D. Lee, J. Kim, W.-J. Moon, J. C. Ye, CollaGAN: Collaborative GAN for Missing Image Data Imputation, in: Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2019), IEEE, 2019, pp. 2487–2496.

[29] M. Kachuee, K. Karkkainen, O. Goldstein, S. Darabi, M. Sarrafzadeh, Generative Imputation and Stochastic Prediction, IEEE Transactions on Pattern Analysis and Machine Intelligence 44 (3) (2020) 1278–1288.

[30] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A. Courville, Improved Training of Wasserstein GANs, in: Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS 2017), Curran Associates Inc., 2017, pp. 5767–5777.

[31] J. T. Hancock, T. M. Khoshgoftaar, Survey on Categorical Data for Neural Networks, Journal of Big Data 7 (1) (2020) 1–41.

[32] M. S. Sajjadi, O. Bachem, M. Lucic, O. Bousquet, S. Gelly, Assessing Generative Models via Precision and Recall, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS 2018), Curran Associates, Inc., 2018, pp. 5234–5243.

[33] S. Santurkar, D. Tsipras, A. Ilyas, A. Madry, How Does Batch Normalization Help Optimization?, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS 2018), Curran Associates, Inc., 2018, pp. 2488–2498.

[34] J. L. Ba, J. R. Kiros, G. E. Hinton, Layer Normalization, in: Proceedings of the 30th International Conference on Neural Information Processing Systems Deep Learning Symposium (NIPS 2016 Deep Learning Symposium), 2016.

[35] M. Bayer, M. Kaufhold, C. Reuter, A survey on data augmentation for text classification, ACM Computing Surveys 55 (7) (2023) 146:1–146:39.

[36] S.-i. Amari, R. Karakida, M. Oizumi, Information Geometry Connecting Wasserstein Distance and Kullback–Leibler Divergence via the Entropy-Relaxed Transportation Problem, Information Geometry 1 (1) (2018) 13–37.

[37] S. Zheng, N. Charoenphakdee, Diffusion models for missing value imputation in tabular data, in: Proceedings of the NeurIPS 2022 First Workshop on Table Representation Learning (TRL 2022), OpenReview.net, 2022.

[38] Y. Ouyang, L. Xie, C. Li, G. Cheng, MissDiff: Training Diffusion Models on Tabular Data with Missing Values, in: Proceedings of the ICML 2023 First Workshop on Structured Probabilistic Inference Generative Modeling (SPIGM 2023), OpenReview.net, 2023.

[39] Y. Liu, T. Ajanthan, H. Husain, V. Nguyen, Self-Supervision Improves Diffusion Models for Tabular Data Imputation, in: Proceedings of the ICLR 2024 First Workshop on Generative Models for Decision Making (GenAI4DM 2024), Amazon Science, 2024.