Bayesian Generative Adversarial Imputation Nets

Francesco Barbara
University of Oxford
Mathematical Institute

Abstract

In this project, we propose, building on the generative adversarial imputation networks (GAIN) framework, a novel algorithm for data imputation called Bayesian generative adversarial imputation network (BGAIN).
This method incorporates prior knowledge on the imputation process which we exploit during training of the generator network.
Moreover, it extends the applicability of GAIN to MAR and MNAR settings (which we show do not have a unique solution for the min-max problem used in GAIN).
gm The main idea is to penalize mroe generators that produce imputed values $\hat{x}$ which disagree with our prior knowledge of what values of $\hat{x}$ are reasonable given observed values $\widetilde{x}$, making the generator converge to a distribution that agrees with our initial intuitions.

1 Introduction

The healthcare domain presents many settings in which data is not missing completely at random (MCAR).
It is indeed very reasonable to assume that missingness in one coordinate is correlated with observed values of other coordinates (for example, a medical professional might decide whether to have a patient undergo a particular exam, based on their available medical records), that the missing rate of a particular variable depends on the latent variable itself, or that missingness is driven by other covariates which are not included in the the available dataset.

In such settings, uniqueness of the solution to the GAIN objective is not guaranteed (not even on a purely theoretical perspective). This can be easily proven by constructing an ad-hoc counter-example. The underlying idea is to start with a fixed pre-imputation matrix $\widetilde{X}$, then define two different conditional densities for the missing data $p^{1-m}(x|\tilde{x})$, $q^{1-m}(x|\tilde{x})$. Because in both scenarios the GAIN algorithm takes the same dataset $\widetilde{X}$ as input, it will produce the same generative distribution in both cases, implying sub-optimality in at least one case (in practice both, unless data is MCAR).

In addition, from a practical perspective, neural networks loss landscapes are highly non-convex, making them susceptible to the risk of getting stuck at local minima.

Given such premises, it is desirable to leverage, in a structured manner, the prior information about the imputation process at our disposal, to guide the generator's neural network towards the global minimum.

A great amount of niche expertise is available in medicine, being able to incorporate such knowledge in our imputations is key to produce sensible generative processes.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

2 Problem Formulation

The problem set-up is similar to that of GAIN, in the sense that there is a random variable $X = (X_1, ..., X_d)$ taking values in $\mathcal{X} = \mathcal{X}_1 \times ... \times \mathcal{X}_d$, and that we are given n independent realizations of $\widetilde{X} = (\widetilde{X}_1, ..., \widetilde{X}_d) \in \widetilde{\mathcal{X}} = \widetilde{\mathcal{X}}_1 \times ... \times \widetilde{\mathcal{X}}_d$ where:

$$\widetilde{X}_i = \begin{cases} X_i, & \text{if } M_i = 1 \\ *, & \text{if } M_i = 0 \end{cases} \tag{1}$$

and M_i represents whether the i-th component of the vector is observed.

In addition, we capture our prior knowledge about the imputation process by defining pairwise covariances for X_i and X_j when one component is observed and the other is not.

$$Cov(X_i \,|\, M_i = 0,\ X_j \,|\, M_j = 1) = \Sigma_{ij}^{(0)} \quad \text{and} \quad \mathrm{E}[X_i \,|\, M_i = 0] = \mu_i^{(0)}$$

This modelling assumption aims to capture our prior believes (ideally acquired through the consultation of experts in the field). For example, if X_i and X_j were two similar exams measuring the same trait (it would indeed be very reasonable in this case to assume that a patient undergoes only one of the two exams), then we would have a correlation close to 1 between X_i and X_j.

We assume that if X_i is not observed, but components X_J for $J \in \{1, ..., d\}$ are, then the following approximation holds:

$$(X_i, X_J)^T \,\Big|\, M_i = 0,\ M_J = \mathbf{1} \quad \approx \quad N(\widetilde{\mu}, \widetilde{\Sigma}) \quad \text{where:}$$

$$\widetilde{\mu} = \begin{pmatrix} \mu_i^{(0)} \\ \mu_J^{(1)} \end{pmatrix} \quad \text{and} \quad \widetilde{\Sigma} = \begin{pmatrix} \Sigma_{ii}^{(0)} & \Sigma_{iJ}^{(0)} \\ \left(\Sigma_{iJ}^{(0)}\right)^T & \Sigma_{JJ}^{(1)} \end{pmatrix}$$

$$\implies X_i \,\Big|\, M_i = 0,\ X_J = x_J,\ M_J = \mathbf{1} \quad \approx \quad N(\mu_c, \sigma_c^2) \quad \text{where:}$$

$$\mu_c = \mu_i^{(0)} + \Sigma_{iJ}^{(0)}\ {\Sigma_{JJ}^{(1)}}^{-1} (x_J - \mu_J^{(1)})$$

$$\sigma_c^2 = \Sigma_{ii}^{(0)} - \Sigma_{iJ}^{(0)}\ {\Sigma_{JJ}^{(1)}}^{-1} (\Sigma_{iJ}^{(0)})^T$$

The above implication follows by standard results on conditional Gaussians.

It is important to remark that, our prior knowledge is represented exclusively in $\Sigma^{(0)}$ and $\mu^{(0)}$ - which have a total dimension of $d^2 + d$ and we have to feed these parameters to the algorithm -, while $\Sigma^{(1)}$ and $\mu^{(1)}$ can be recovered from the observed data. A more detailed explanation will be given in the next section.

Suppose now that a certain value $\hat{X}_i$ is imputed. Then, under the above approximation, the probability density of $\hat{X}_i$ is:

$$q(\hat{x}_i) = \frac{1}{\sqrt{2\pi\sigma_c^2}} exp(-\frac{(\hat{x}_i - \mu_c)^2}{2\sigma_c^2})$$

As it will become clearer in the next section, the aim of the 'Gaussian trick' is not to provide a good approximation of the generative process $P(X|\widetilde{X})$ itself, but rather to penalize imputed values $\hat{x}$ which disagree with our prior knowledge of what values of $\hat{x}$ are reasonable given observed values $\widetilde{x}$, making the generator converge to a distribution that agrees with our initial intuitions.

3 Bayesian Generative Adversarial Imputation Nets (BGAIN)

We adopt the same training scheme as in GAIN for what concerns the discriminator, using loss $\mathcal{L}_D : \{0,1\}^d \times [0,1]^d \times \{0,1\}^d \to \mathbb{R}$ defined by:

$$\mathcal{L}_D(\mathbf{m}, \hat{\mathbf{m}}, \mathbf{b}) = - \sum_{i:b_i=0} [m_i log(\hat{m}_i) + (1 - m_i) log(1 - \hat{m}_i)]$$

The novelty introduced, using the formalism proposed in the previous section, is to add a third loss term $\mathcal{L}_P$, which penalizes imputed values distant from our prior.
$\mathcal{L}_P : \mathbb{R}^d \times \mathbb{R}^d \times \{0,1\}^d \to \mathbb{R}$ is defined as:

$$\mathcal{L}_P(\hat{\mathbf{x}}, \widetilde{\mathbf{x}}, \mathbf{m}) = - \sum_{i=1}^{d} (1 - m_i) q(\hat{x}_i \,|\, \mu_c, \, \sigma_c^2)$$

Note that dependence on $\widetilde{\mathbf{x}}$ is implicit through μ_c and σ_c^2.

We can see that $\mathcal{L}_P$ only considers coordinates that have been imputed. For such coordinates, it will reward the generator if it has chosen values that can realistically come from the conditional Gaussian density specified earlier, and penalize it otherwise.

The other two losses $\mathcal{L}_G$ and $\mathcal{L}_M$, capturing respectively the ability of D to recognise imputed values and the dissimilarity between generated features and their real counterparts, are defined as in GAIN:

$$\mathcal{L}_G(\mathbf{m}, \hat{\mathbf{m}}, \mathbf{b}) = - \sum_{i:b_i=0} (1 - m_i) log(\hat{m}_i)$$

$$\mathcal{L}_M(\hat{\mathbf{x}}, \widetilde{\mathbf{x}}, \mathbf{m}) = \sum_{i=1}^{d} m_i (\hat{x}_i - \widetilde{x}_i)^2$$

Before jumping to the complete BGAIN algorithm, we should provide details on how to compute $\Sigma^{(1)}$ and $\mu^{(1)}$ from a dataset $\{\widetilde{\mathbf{x}}(k), \mathbf{m}(k)\}_{k=1}^n$. Such computation, highlighted in **Algorithm 1**, can be done at initialization time, and will allow us to compute μ_c and σ_c^2 on the spot during training time.

Algorithm 1 Computation of $\Sigma^{(1)}$ and $\mu^{(1)}$

1: **Inputs**: Dataset $\{\widetilde{\mathbf{x}}(j), \mathbf{m}(k)\}_{j=1}^n$
2: Center the data matrix $\hat{\mathbf{X}}$ by subtracting column means $\mu^{(1)}$
3: **for** each $(i, \, j) \in \{1, ..., d\}^2$ **do**
4: $\quad S_i = \{k \leq n : \, m(k)_i = 1\}$
5: $\quad S_j = \{k \leq n : \, m(k)_j = 1\}$
6: $\quad \left(\Sigma^{(1)}\right)_{ij} = \frac{1}{|S_i \cap S_j|} \sum_{k \in S_i \cap S_j} (x(k))_i (x(k))_j$
7: **Return** $\Sigma^{(1)}$, $\mu^{(1)}$

Algorithm 2 BGAIN

Inputs: Dataset $\{\widetilde{\mathbf{x}}(j), \mathbf{m}(j)\}_{j=1}^{n}$, Priors $\Sigma^{(0)}$ and $\mu^{(0)}$, Hyperparameters α, β, k_D, k_G to be chosen by Cross-Validation
for each column $\widetilde{X}_{\bullet j}$ $j \leq d$ **do**
    Calculate the mean $\mu_j^{(1)}$ and standard deviation $\sigma_j^{(1)}$ of $\widetilde{X}_{\bullet j}$ discarding missing values
    Normalize the column: $\widetilde{X}_{\bullet j} \leftarrow (\widetilde{X}_{\bullet j} - \mu_j^{(1)})/\sigma_j^{(1)}$ ▷ good practice for NN training
Calculate $\Sigma^{(1)}$ using Algorithm 1
Readjust $\Sigma^{(0)}$ and $\mu^{(0)}$ to account for normalization of features
for each $(i,\, j) \in \{1, ..., d\}^2$ **do**
    $\Sigma_{ij}^{(0)} \leftarrow \Sigma_{ij}^{(0)}/(\sigma_i^{(1)}\sigma_j^{(1)})$
for each $i \in \{1, ..., d\}$ **do**
    $\mu_i^{(0)} \leftarrow (\mu_i^{(0)} - \mu_i^{(1)})/\sigma_i^{(1)}$

while Not convergence **do**
    Discriminator Training ▷ same as GAIN
    Sample a batch of size k_D $\{\widetilde{\mathbf{x}}(j), \mathbf{m}(j)\}_{j=1}^{k_D}$ and corresponding hints
    Make a gradient descent step using the following gradient:
    $\nabla_D \left(\sum_{j=1}^{k_D} \mathcal{L}_D(\mathbf{m}(j), \hat{\mathbf{m}}(j), \mathbf{b}(j)) \right)$

    Generator Training
    Sample a batch of size k_G $\{\widetilde{\mathbf{x}}(j), \mathbf{m}(j)\}_{j=1}^{k_G}$ and corresponding hints
    $\mathcal{L} \leftarrow 0$ ▷ Initializing the total loss
    for $j \in \{1, ..., k_G\}$ **do**
        $\mathcal{L} \leftarrow \mathcal{L} + \mathcal{L}_G(\mathbf{m}(j), \hat{\mathbf{m}}(j), \mathbf{b}(j))$
        $\mathcal{L} \leftarrow \mathcal{L} + \alpha \cdot \mathcal{L}_M(\hat{\mathbf{x}}(j), \widetilde{\mathbf{x}}(j), \mathbf{m}(j))$

        Calculate σ_c^2 and μ_c and use them to calculate $q(\hat{x}(j)_i \,|\, \mu_c,\, \sigma_c^2)$
        $\mathcal{L}_P(\hat{\mathbf{x}}(j), \widetilde{\mathbf{x}}(j), \mathbf{m}(j)) \leftarrow -\sum_{i=1}^{d} (1 - m(j)_i) q((\hat{x}(j))_i \,|\, \mu_c,\, \sigma_c^2)$
        $\mathcal{L} \leftarrow \mathcal{L} + \beta \cdot \mathcal{L}_P(\hat{\mathbf{x}}(j), \widetilde{\mathbf{x}}(j), \mathbf{m}(j))$
    Make a gradient descent step using $\nabla_G(\mathcal{L})$

A remark worth making, is that even though the algorithm requires us to specify a $d \times d$ matrix $\Sigma^{(0)}$ and a $d \times 1$ vector $\mu^{(0)}$, especially in high-dimensional cases, we can use sparse matrices where we only specify relevant non-zero covariances. This would still ensure some sort of regularization towards distributions with the desired properties, at a fraction of the cost at initialization time.

A second technical remark is that, in my TensorFlow implementation, the step at line 25 is done much more efficiently and for each observation you only need to do a single matrix multiplication to obtain μ_c and σ_c^2 without the need to perform an expensive matrix inversion each time (which is done only once at initialization time).

4 Possible Extensions and Limitations

The exact same idea, could be potentially used to improve the first imputation step of the MissForest algorithm, in which initially data is imputed using the observed means.

It is also noteworthy that, even though the BGAIN algorithm also works with discrete ordered variables (you would be just evaluating the Gaussian density at finitely many points), it does not apply to categorical settings, limiting its applicability.

Another cause of concern at training time, would be the choice of hyperparameter β which determines the effect of the new loss introduced $\mathcal{L}_P$. If β is too high (compared to the other two terms) it will happen that the generator will output exclusively values around $\mu^{(0)}$, therefore you will end up using

the initial approximation for the imputation which is undesirable (the main idea is to use deep learning for learning a generative model and penalize points far away from the prior, not to use the prior as the imputation mechanism). As a consequence, a careful choice of α and β through cross-validation is recommended.