PROMISSING: PRUNING MISSING VALUES IN NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

While data are the primary fuel for machine learning models, they often suffer from missing values, especially when collected in real-world scenarios. However, many off-the-shelf machine learning models, including artificial neural network models, are unable to handle these missing values directly. Therefore, extra data preprocessing and curation steps, such as data imputation, are inevitable before learning and prediction processes. In this study, we propose a simple and intuitive yet effective method for pruning missing values (PROMISSING) during learning and inference steps in neural networks. In this method, there is no need to remove or impute the missing values; instead, the missing values are treated as a new source of information (representing what we do not know). Our experiments on simulated data, several classification and regression benchmarks, and a multi-modal clinical dataset show that PROMISSING results in similar classification performance compared to various imputation techniques. In addition, our experiments show models trained using PROMISSING techniques are becoming less decisive in their predictions when facing incomplete samples with many unknowns. This finding hopefully advances machine learning models from being pure predicting machines to more realistic thinkers that can also say "I do not know" when facing incomplete sources of information.

1 INTRODUCTION

Missing and incomplete data are abundant in real-world problems; however, the learning and inference procedures in machine learning (ML) models highly rely on high-quality and complete data. Therefore, it is necessary to develop new methods to deal with data imperfections in rugged environments. Currently, the most popular way to deal with imperfect data is to impute the missing values. However, if we consider the learning and inference procedures in our brain as a role model for ML algorithms, data imputation barely follows the natural principles of incomplete data processing in our brain. This is because the imputation is generally based on using a heuristic for *replacing* missing values. Our brain does not impute incomplete sensory information but instead uses its incompleteness as a separate source of information for decision making. For example, by only hearing the rain we can estimate how hard it is raining, and we do not necessarily need to receive visual information. Instead, we direct our attention more toward our auditory inputs to decide whether to go out with an umbrella. In addition, the more we miss sensory information, the more cautious we get in decision-making. That is why we are more careful in darker environments.

Neural networks (NNs) are brain-inspired algorithms that are very popular these days (under the name of deep learning) for learning complex relationships between inputs and target variables. However, they are in principle unable to handle incomplete data with missing values. They mainly rely on matrix operations which cannot operate on not-a-number (NaN) values. Only one NaN in a dataset impairs the forward propagation in a network. There are three solutions to this problem (García-Laencina et al., 2010): i) removing samples or features with missing values, ii) imputing the missing values, and iii) modeling the incomplete data. Removing the samples with missing values can be very costly, especially in small-sample size datasets. For example, data collection in clinical applications is an expensive procedure in terms of time, finance, and patient burden. Moreover, removing even a few samples from small datasets can affect negatively the generalization performance of the final model. Removing informative features with missing values is also compensated with lower model performance. Therefore, filling the information gaps is inevitable.

There are various techniques for data imputation, ranging from simply imputing missing values with a constant to more sophisticated ML-based imputation approaches (Little & Rubin, 2019; García-Laencina et al., 2010). One can categorize the most common techniques into three main categories: i) constant imputation, ii) regression-based imputation, and iii) ML-based imputation. In constant imputation, the missing values are replaced with a constant, *e.g.*, zeros or mean/median of features. While being very simple, it has been shown that constant imputation is only effective when the missing features are not informative (Josse et al., 2019). In the regression-based imputation, a linear or non-linear regression model is derived to predict the missing values. This method can be used to impute a single or multiple features (multiple imputation). The most popular regression-based imputation is Multiple Imputation by Chained Equations (MICE) (Van Buuren & Groothuis-Oudshoorn, 2011; Azur et al., 2011). In MICE, an iterative procedure of predicting missing values and re-training regressors with updated predictions is performed for a limited number of cycles. The central assumption behind the MICE approach is that the missing values are missed at random (MAR) (Azur et al., 2011), *i.e.*, the probability that a feature is missing depends only on observed values and not missing values.[1] Applying MICE can result in biased estimations if this assumption is not satisfied (Azur et al., 2011). Another critical limitation of regression-based imputation is its high computational complexity (Caiafa et al., 2021). If we do not know which feature will be missed at the test time, for d features, we need to train d different regression models in advance. In the ML-based imputation, an ML algorithm such as a K-nearest neighbor (KNN), random forest (Xia et al., 2017), neural network (Bengio & Gingras, 1996), or generative model (Yoon et al., 2018) is used for data imputation (see García-Laencina et al. (2010) and Jäger et al. (2021) for some examples).

As an alternative solution to data imputation, one can use the elegance of probabilistic modeling to model the incomplete data under certain assumptions. One seminal work in this direction is presented by Ghahramani & Jordan (1994), where a Gaussian Mixture Model (GMM) is used to estimate the joint density function on incomplete data using an Expectation-Maximization (EM) algorithm. This approach is later adopted and extended to logistic regression (Williams et al., 2005), Gaussian processes, support vector machines (Smola et al., 2005), and multi-class non-linear classification (Liao et al., 2007). However, despite their good performance on small-size datasets, their application remained limited on big and high-dimensional data due to the high computational complexity (Caiafa et al., 2021). To overcome this issue, Caiafa et al. (2021) proposed a sparse dictionary learning algorithm that is trained end-to-end, and simultaneously learns the parameters of the classifier and sparse dictionary representation. Le Morvan et al. (2020) proposed NeuMiss, a neural-network architecture that uses a differentiable imputation procedure in a impute-then-regress scheme (Morvan et al., 2021). A notable feature of NeuMiss is its robustness to MNAR data. Recently, Smieja et al. (2018) proposed a modified neuron structure that uses GMM with a diagonal covariance matrix (assuming MAR) to estimate the density of missing data. GMM parameters are learned with other network parameters. Conveniently, it handles missing values in the first layer of the network, and the rest of the architecture remains unchanged. Elsewhere, Nowicki et al. (2016) proposed a new neural network architecture based on rough set theory (Pawlak, 1998) for learning from imperfect data. It is fascinating that this method can say "I do not know" when a large portion of input values are missing, unlike traditional models trained on imputed data that may predict definite outcomes even on completely unmeasured samples, *i.e.*, they run in the absolute darkness.

In this work, we attack the problem of modeling incomplete data using artificial neural networks without data imputation. We propose a simple technique for pruning missing values (PROMISSING) in which the effect of missing values on the activation of a neuron is neutralized. In this strategy, a missing value is not replaced by arbitrary values (*e.g.,* through imputation); it is naturally considered a missing piece of the puzzle; *we learn a problem-specific numerical representation for unknowns*. The key feature of PROMISSING is its simplicity; it is plug-and-play; it deals with missing values in the first layer of the network without the need to change anything in the rest of network architecture or the optimization process. PROMISSING in its original form does not add extra parameters to the network, and its computational overhead remains negligible. Our experiments on simulated data and several classification/regression problems show that the proposed pruning method does not negatively affect the model accuracy and provides competitive results compared to several data imputation techniques. In a clinical application, making prognostic predictions for patients with a psychotic disorder, we demonstrate an application of PROMISSING on a multi-modal

[1] See Ghahramani & Jordan (1995) and Schafer & Graham (2002) for definition and discussion on missing value mechanisms including completely at random (MCAR), at random (MAR), and not at random (MNAR).

clinical dataset. We demonstrate how the NN model trained using PROMISSING becomes indecisive when facing many unknowns. Furthermore, we show a side application of PROMISSING for counterfactual interpretation (Mothilal et al., 2020) of NNs decisions that can be valuable in clinics.

2 METHODS

Let $\boldsymbol{x} \in \mathbb{R}^p$ represent a vector of an input sample with p features. We assume that the features in $\boldsymbol{x}$ are divided into two sets of q observed $\boldsymbol{x}^o \in \mathbb{R}^q$ and r missing features $\boldsymbol{x}^m \in \mathbb{R}^r$ (where $p = q + r$). We do not put any assumption on the pattern of missing values in $\boldsymbol{x}$, and this pattern can be different from one sample to another. Then, the activation of the kth ($k \in \{1, 2, \ldots, s\}$) neuron in the first hidden layer of an NN is defined as:

$$a^{(k)} = \sum_{x_i \in \boldsymbol{x}^o} x_i w_i^{(k)} + \sum_{x_j \in \boldsymbol{x}^m} x_j w_j^{(k)} + b^{(k)}. \tag{1}$$

This activation cannot be computed unless the values in $\boldsymbol{x}^m$ are imputed with real numbers. Here, in PROMISSING, we propose to alternatively replace the missing values with a *neutralizer* that 1) prunes the missing values from inputs of a neuron, 2) neutralizes its effect on the neuron's activation. A missing value $x_j \in \boldsymbol{x}^m$ is replaced with its corresponding neutralizer $u_j^{(k)}$ at the kth neuron, where:

$$u_j^{(k)} = \frac{-b^{(k)}}{p w_j^{(k)}}. \tag{2}$$

The value of a neutralizer depends on its corresponding weight ($w_j^{(k)}$), the bias of the corresponding neuron ($b^{(k)}$), and the number of features (p); thus, it can be computed on the fly during the training or inference procedures. A small value is added to weights before computing neutralizers to avoid division by zero. Inserting the neutralizer into Eq. 1, the activation of the kth neuron is rewritten as:

$$a^{(k)} = \sum_{x_i \in \boldsymbol{x}^o} x_i w_i^{(k)} + \frac{q b^{(k)}}{p}, \tag{3}$$

in which the effect of weights of missing values on the activation of the neuron is eliminated, and the neuron's bias is reduced by a factor of r/p. If all input values for a specific sample are missing ($q = 0$ and $\boldsymbol{x}^o = \varnothing$), the outcoming activation will be 0, *i.e.*, the neuron is completely neutralized. We should emphasize that, when using PROMISSING, the user does not need to apply any change to the input vectors, and the missing values (generally represented as `nans` in the input matrix) are fed directly to the network. After the training procedure, we eventually learn $\boldsymbol{U} \in \mathbb{R}^{s \times p}$ a matrix representation for unknowns (or metaphorically the dark matter):

$$u_j^{*(k)} = \frac{-b^{*(k)}}{p w_j^{*(k)}} \qquad j \in \{1, 2, \ldots, p\}, \quad k \in \{1, 2, \ldots, s\}, \tag{4}$$

where b^* and w^* are representing the final learned bias and weight. At the prediction stage, a missing value at jth input feature will be replaced with its corresponding neutralizer from $\boldsymbol{U}$. It is worth emphasizing that the missing values are replaced with different neutralizers at different neurons; therefore, it cannot be considered a constant imputation. In fact, each neuron perceives differently a missing value in the input space. Metaphorically, the neurons can be seen as blind men in the parable of "the blind men and an elephant" [2] when facing unknowns. Furthermore, it is different from regression-based imputation and model-based approaches in the sense that a missing value in a specific feature is not inferred from other observed features, or the distribution of observed values; *i.e.*, in PROMISSING unknowns remain unknowns. Therefore, the MAR assumption in the multiple imputation and model-based approaches is relaxed. This relaxation may result in more flexibility of PROMISSING when the patterns of missing values in inputs follow the MNAR mechanism.

One possible drawback of using PROMISSING is in high-dimensional input spaces and when the number of missing values is large, *i.e.*, when $p \to \infty$ and $r \gg q$. In this case, the neuron will undershoot; hence the effect of few non-missing values are ignored. [3] To address this problem, we

[2] See `https://en.wikipedia.org/wiki/Blind_men_and_an_elephant` for the parable.

[3] Note that this again might be considered as an advantage in some applications.

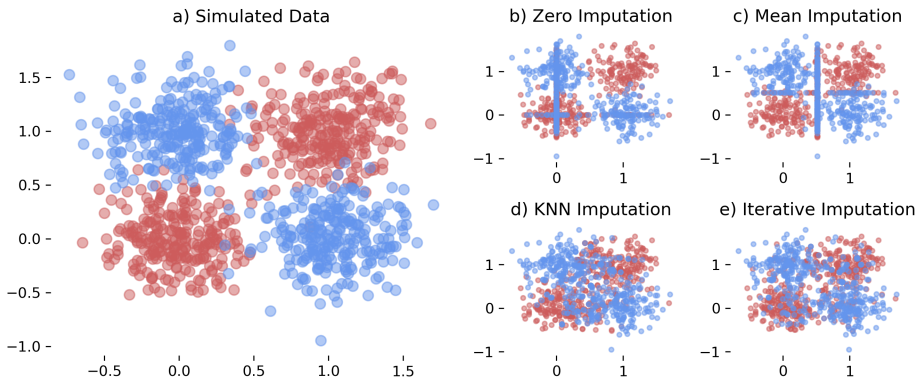


Figure 1: a) The simulated XOR data. The effect of b) zero, c) mean, d) KNN, and e) iterative imputations on simulated data.

propose a *modified* version of PROMISSING (mPROMISSING) in which the effect of large r can be compensated with a *compensatory* weight, w_c. The compensatory weight receives a fixed input of q/p for a specific sample; thus, the activation of the neuron will change to:

$$a^{(k)} = \sum_{x_i \in \boldsymbol{x}^o} x_i w_i^{(k)} + \frac{q(b^{(k)} + w_c^{(k)})}{p}. \tag{5}$$

$w_c^{(k)}$ is learned alongside the rest of the network parameters in the optimization process. On training data with few missing values, data augmentation (*e.g.*, by simulating different patterns and size of missing values) is advisable to ensure the sensibility of the learned compensatory weight. Since the input for this weight (q/p) is computed at the run time, no modification on input vectors is required.

The proposed PROMISSING approach is straightforward to implement and use. It can be incorporated into the current implementations of different types of NN layers by adding/modifying few lines of code. We have implemented a `nanDense` layer, inheriting from Keras (Chollet et al., 2015) `Dense` layer, using the PROMISSING and mPROMISSING techniques (see appendix A.1). The `nanDense` layer can be directly imported and used with any Keras model. In its general usage, the `nanDense` layer is only used for the first layer of an NN to handle missing values in inputs, unless we expect some missing values in the intermediate layers.

3 Experiments and Results

In a set of three experiments, 1) we investigate the convergence behavior of PROMISSING and the learned representations of unknowns in a simulation study; 2) in a large experiment on real data, we benchmark the reliability of PROMISSING in several classification and regression tasks, and 3) on a multi-modal clinical dataset, we demonstrate an application of PROMISSING on an open problem of psychosis prognosis prediction.

3.1 Simulated Data

We have conducted a simple analysis of synthesized data to better understand the behaviors of PROMISSING. For data simulation, we have simulated the XOR problem with 1000 samples. Gaussian noise with a variance of 0.25 is added to data as depicted in Fig. 1a. To simulate the missing data, we randomly removed one or two features (in an MCAR scheme) from 50% of samples. Fig. 1b-e illustrate the effect of applying different data imputation methods on the data. We used scikit-learn (Pedregosa et al., 2011) for implementing different data imputation schemes.[4] For the KNN imputer, the missing values are imputed with the average of 5 nearest neighbors. In the iterative imputation (that corresponds to MICE imputation), a random forest regressor is used for regressing missing values. As shown in the figure, in all cases, imputation results in misrepresentation of samples with missing values, and consequently losing the sharp boundaries between categories.

[4]Scikit-learn 0.24.2 with Numpy 1.19.5.

Figure 2: Comparison between learning curve of the full model with PROMISSING and mPROMISSING on the simulated data, when tested on a test set a) without missing values, and b) with missing values.

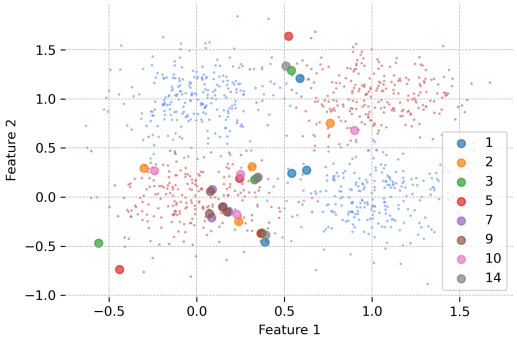


Figure 3: Pattern of neutralizers in $\boldsymbol{U}$ across two input features and the four neurons. The markers with the same color show the neutralizer values for each of four neurons in a specific run (legends) of the pipeline. The blue and red dots in the background are representing the simulated data.

To evaluate the qualities of (m)PROMISSING, we used a simple fully connected NN architecture with four tangent-hyperbolic neurons in the hidden layer and a single sigmoid neuron in the output layer. This simple architecture can reach the performance of the Bayes optimal classifier (AUC=0.99 ± 0.01) on simulated data (without missing values). We randomly divided the simulated data into training and test sets with 500 samples in each set. We trained the network in two different scenarios i) on the full training set without missing data and ii) on the training set with missing values using (m)PROMISSING methods. The pipeline of 1) data simulation, 2) random data corruption with missing values, 3) randomly splitting data into the training and test set, and 4) training the model is repeated 50 times. We have fixed the random number generator seed in each repetition (randomly selected from $[0, 10000]$) to ensure a fair comparison between the three resulting models. A stochastic gradient descent (SGD) optimizer is used to minimize a binary cross-entropy loss function. At the prediction phase, we tested the models on test sets i) without and ii) with missing values. Fig. 2a shows the area under the ROC curve (AUC) of models tested on test sets without missing values as the number of epochs progresses. The NN model trained on full data converges much faster (~ 300 epochs earlier); however, the (m)PROMISSING models are eventually reaching the performance of the Bayes classifier, although they are trained on corrupted data. We have not empirically investigated this delay, but we expect it is modulated by the amount of data corruption. For example, if we increase the proportion of missing values from 50% to 90%, then the (m)PROMISSING models receive fewer complete samples during the training, thus converge slower.

Fig. 2b shows the learning curve on test sets with missing values. For the plain NN model (that cannot handle missing values *per se*), a KNN imputer is used for data imputation. Similar to the previous scenario, the (m)PROMISSING methods need more training time to reach the same performance as the full model; however, their AUCs eventually surpass the performance of the NN model trained on complete data. These observations are interesting in two aspects, i) the (m)PROMISSING approaches enable us to learn equivalent models from the corrupted data; ii) the learned representations for the unknowns are at least as informative as imputed data.

To better understand the learned representation of unknowns $\boldsymbol{U}$, we have visualized the elements in $\boldsymbol{U}$ across 50 simulation runs (see Fig. 8 in appendix A.2). Since $\boldsymbol{U}$ is derived from the network parameters, due to the stochastic nature of parameter initialization and optimization, it may end

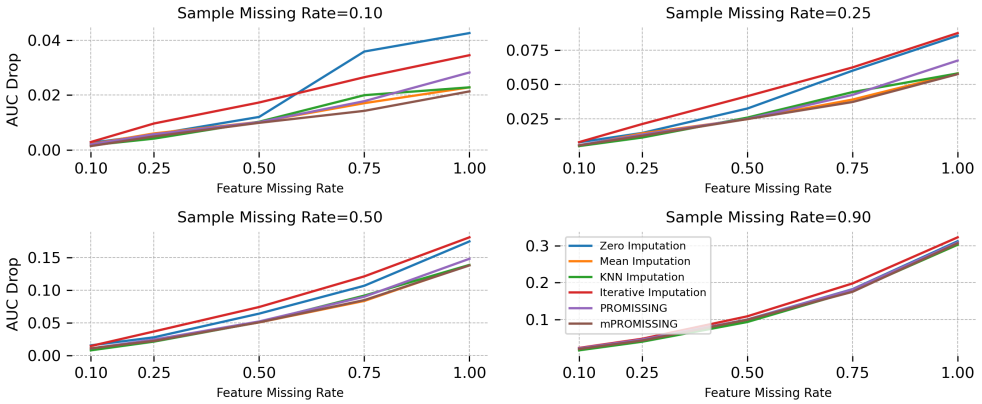


Figure 4: Comparing the AUC drops with respect to the full model across four different imputation methods and (m)PROMISSING in classification tasks.

up with different values in each run (local minima). Our visual inspection shows that the network finds four main groups of solutions for representing unknowns. Fig. 3 depicts elements in $\boldsymbol{U}$ for 8 selected runs that represent these four groups. In this case, $\boldsymbol{U}$ has eight elements (two inputs for four neurons). Markers with similar colors represent the neutralizer values for two features (axes) and four neurons in a specific run (legends). The neutralizers in the first group (7 and 9) are roughly zero imputers in which the missing values are imputed with close to (but not absolutely) zero values. These are sub-optimal solutions that happen only in 2 runs out of 50. In the other 48 runs, the models seem to perform more than a simple imputer. In another minority group, run 1 and 14, the model turns to a mean imputer for the first feature, but it learns diverse representation for the second feature; the neurons see unknowns as borderline samples. In another minority group (represented by run 3 and 5), some neurons see unknowns as samples with extreme values (outside or close to the outer range of inputs). Run 2 and 10 represent the majority group of solutions in which each neuron has its unique perception of unknowns.

3.2 OpenML Data

In the second set of experiments, we benchmarked (m)PROMISSING alongside several imputation techniques on a range of publicly available classification and regression datasets from OpenML (Vanschoren et al., 2014). We used the same classification regression datasets as in Jäger et al. (2021) (see Table 2 and Table 3 in appendix A.3). Here, we use a simple fully connected architecture with two hidden layers, with $p/2$ ReLU neurons in the first and two neurons in the second hidden layer; so the architecture changes slightly from one dataset to another depending on the number of features. A single output neuron with a sigmoid or linear transfer function is used for output layers in classifiers or regressors, respectively. Our quick experiments on benchmark datasets show that this simple architecture provides competitive results compared to random forest classifiers and regressors (see Fig 9 in appendix A.3). We excluded three classification and three regression datasets from the rest of our analyses due to the poor performance of baseline models on the complete data (see appendix A.3 for more details). We used an MCAR scheme to simulate different ratios of samples (0.1, 0.25, 0.5, 0.9) and features (0.1, 0.25, 0.5, 0.75, 1) with missing values across datasets. An SGD optimizer for 100 epochs with a mini-batch size of 10 samples is used in the training process to minimize a binary cross-entropy or a mean squared error (MSE) in the classification or regression scenario. The performances of different methods are evaluated in a 2-fold cross-validation scheme computing the AUC and standardized MSE (SMSE) metrics. The whole experiment pipeline, including 1) simulating missing values, 2) data imputation (this step does not apply to (m)PROMISSING), 3) data standardization, 4) model training and evaluation, is repeated ten times with a fixed random number generator seed in each run. When imputing data, the default imputer settings are applied.

Fig. 4 compares the performance of different data imputation methods with (m)PROMISSING on the classification tasks for different missing sample ratios (in each panel) and missing feature ratios (x-axes). The AUC drops with respect to the full model (trained on complete data) are averaged

across different datasets and ten runs. The (m)PROMISSING shows competitive performance compared to the better performing imputers, *i.e.*, the mean and KNN imputers. Especially, mPROMISSING performs slightly better when the ratio of missing features is increasing. A similar overall pattern can be observed when comparing the increase in the SMSE in the regression tasks (see Fig. 10 in appendix A.3). In summary, these results show that (m)PROMISSING can compete with data imputation approaches without the need for filling the unknowns.

3.3 Clinical Application: Psychosis Prognosis Prediction

Missing data are seemingly ubiquitous in the healthcare domain. In general, clinical datasets contain multi-modal data that are collected through different measurement tools, including patient questionnaires, medical reports and instruments, biological tests, and imaging data. Each modality may represent a different aspect of a disease or treatment outcome. Therefore, employing smart modality fusion techniques is inevitable to best benefit from these rich data. However, this goal is occluded because these data are collected through different sources and often include missing variables or even missing modalities. In this section, we demonstrate an application of PROMISSING in multi-modal data fusion and counterfactual model interpretation for psychosis prognosis prediction (PPP).

3.3.1 OPTiMiSE Dataset

We used the OPTiMiSE dataset (Kahn et al., 2018), an antipsychotic three-phase switching study. This dataset contains 495 patients with schizophrenia, schizophreniform, or schizoaffective disorder according to DSM-IV diagnosis standards. The patients received different medications in a three-phase study design. We used the data from the first phase of the study in which the patients were treated with amisulpride for four weeks. Only 376 patients who finished the first phase are used in our experiments. In a PPP framework, we aimed to predict the probability of patients' symptomatic remission based on the Positive and Negative Syndrome Scale (PANSS) as defined by consensus criteria (Andreasen et al., 2005). We used 12 different data modalities in our experiments (see Table 1). In total, 13% values in the dataset are missing. These missing values include complete or partial missed modalities. The missing values are scattered across patients and measures; somehow, if we remove all subjects with missing values only 17 patients are left. This observation substantiates, even more, the importance of missing data management in clinical datasets. We have applied a minimal preprocessing pipeline on the data before the modeling stage. The input data contains binary, categorical, and continuous variables. We used 0/1 and one-hot encoding for binary and categorical variables, respectively. All continuous variables are standardized before feeding them to the model. Considering the unbalanced class distributions (67% of patients get remitted), in each fold of K-fold cross-validation, we resampled the minority class samples in the training data to match its size to the majority class.

Table 1: Data modalities in the OPTiMiSE dataset.

	Modality	Measures Num.	Samples with NaNs	Percentage of NaNs	Description
1	Demographics	20	300	16%	Socio-demographic features
2	Diagnosis	6	45	6%	Illness related features
3	Lifestyle	7	421	21%	Use of substances like drugs and alcohol
4	Somatic	11	92	13%	Psychical examination
5	Treatment	1	67	14%	Average dosage of medication
6	MINI	67	47	8%	Psychiatric comorbidity
7	Cytokines	34	100	20%	Proteins important in cell signaling
8	PANSS	30	42	8%	Positive and Negative Syndrome Scale
9	PSP	5	57	11%	Personal and Social Performance Scale
10	CGI	1	46	9%	Clinical Global Impression
11	CDSS	9	53	11%	Measurement scale about depression
12	SWN	20	81	16%	Subjective Well-Being Under Neuroleptic
	Summary	211	-	13%	-

3.3.2 A Multi-Modal Model for PPP

Inspired by Huang et al. (2020), we used a multi-modal NN architecture with middle and late information fusion. As depicted in Fig. 5, our model has five layers: input, representation learning, modality-specific classification, fusion, and classification. The network receives the data from M different modalities in the input layer. Since input samples may contain missing values, we use `nanDense` layers right after the input layer in the representation learning phase. These layers transfer the raw inputs to an intermediate representations. We roughly (without any optimization and solely based on input feature size) fixed the number of neurons in this layer for each modality (Demographics=10, Diagnosis=10, Lifestyle=5, Somatic=5, Treatment=2, MINI=10, Cytokines=10, PANSS=10, PSP=5, CGI=2, CDSS=5, SWN=5). In the modality-specific classification layer, we use 2 Softmax neurons to classify the modality-specific representations into target classes, *i.e.*, whether a patient remits given a certain amount of medication after four weeks. These decisions are merged with middle representation and then fed to the fusion layer. A trivial ReLU `Dense` layer

with five neurons is used for the fusion layer. Finally, two Softmax neurons are used to classify the fused data into target classes. Dropout layers with a probability of 0.1 are applied before any weighted layer. An Adam (Kingma & Ba, 2014) optimizer (with learning rate of 0.0003) is used to minimize the total categorical cross-entropy loss across classification layers.

3.3.3 Evaluation and Comparison

We first compared (m)PROMISSING with two alternative strategies: i) removing features with missing values and ii) data imputation. In the first case, we first removed subjects that completely missed one data modality; then, we removed features containing missing values. This procedure ended up losing 67 subjects and 78% of features including two modalities. In the second case, we used a KNN to impute the missing data. Note that, due to binary and categorical features, many imputation techniques (such as mean and iterative imputer) cannot be directly applied to our data. Even in the KNN case, we needed to set the number of neighbors equal to one to avoid float numbers for the categorical features. A 10-fold cross-validation scheme is employed for model evaluations, and all the experiments are repeated ten times to evaluate the variability in performances. The results support our conclusion in Sec. 3.2 that (m)PROMISSING performs as well as data imputation. PROMISSING and mPROMISSING result in AUCs of 0.67 ± 0.02 and 0.66 ± 0.02 that are on a par with the KNN imputer 0.66 ± 0.02. However, as is expected, removing missing features resulted in an impaired model performance of 0.61 ± 0.02. The similar performance of (m)PROMISSING and data imputation despite their completely different strategies to deal with missing values raises two questions: "Do the resulting models behave similarly too?", "If not, how are they different?". Craving to answer these questions, we set up the second set of experiments on the OPTiMiSe data.

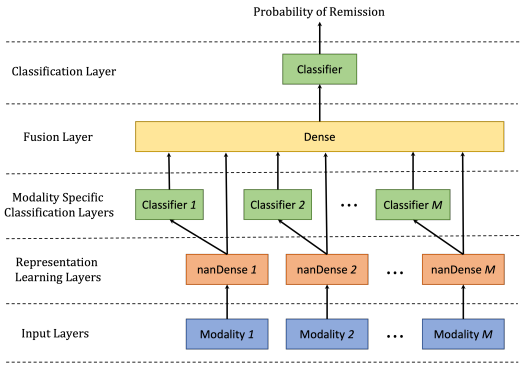


Figure 5: The NN architecture with middle and late information fusion for psychosis prognosis prediction on multi-modal data.

In the next experiment, we left all the 17 patients with complete data in the test set and used the other 359 patients (with missing data) for training. Here, to fully use the capabilities of (m)PROMISSING models, we have augmented the training data by removing different subsets of modalities from data. Having $M = 12$ modalities and $n = 470$ samples (after balancing the training data) in the training set, this procedure resulted in $1,919,010$ samples ($\sum_{m=0}^{M-1} \binom{M}{m} \times n$). We trained three models, one using KNN imputation and two using (m)PROMISSING. Then, we have used a particular procedure for testing these models on the test set. In 100 repetitions, we have removed the modalities one by one and in random order. Fig. 6 shows the trajectories of predictions for 17 test subjects across three models. The trajectories show the predicted probability of remission when the model has access to less information about the patients (from complete information on the left to no information on the right). Note that it is impossible to remove all modalities in the KNN imputation case because it impairs the imputation mechanism. The interesting observation is the difference between the prediction behavior of models when they have less information available. The model trained and tested

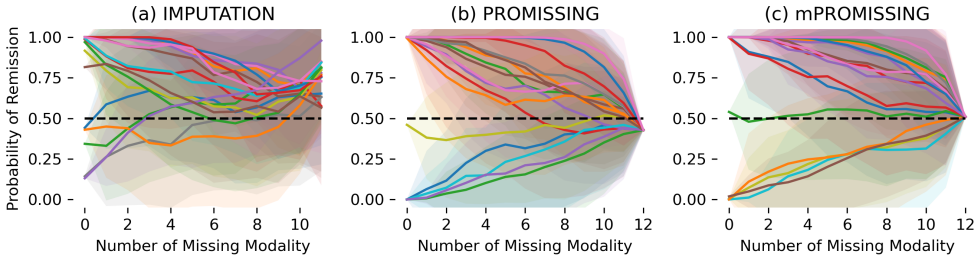


Figure 6: The trajectories of model predictions for 17 patients with information loss when using a) data imputation, b) PROMISSING, and c) mRPOMISSING methods to handle missing data.

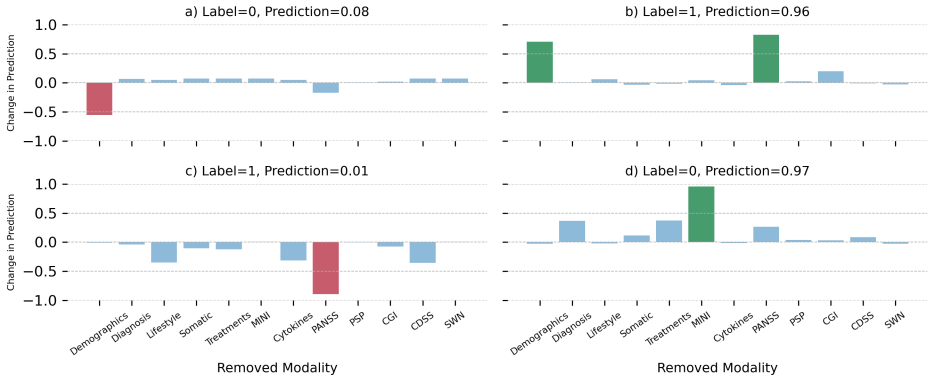


Figure 7: Counterfactual interpretation of model decisions using mPROMISSING for a) true negative, b) true positive, c) false negative, and d) false positive examples in model predictions.

on imputed data always reacts to inputs even in an unknown environment. While the predictions of m(PROMISSING) models monotonically converge to 0.5, *i.e.*, the models become indecisive in an unknown environment. As expected, the predictions of the PROMISSING model are slightly below the chance when the number of missing values increases, but this divergence is corrected in mPROMISSING. This feature is clinical in delicate applications of ML in the medical domain in which the decisions made by machines can significantly affect the quality of life of patients.

One possible application of PROMISSING is in counterfactual interpretation (CI) (Mothilal et al., 2020) of model decisions. CI is an effective method for the causal interpretation of complex models such as NNs. PROMISSING facilitates the usage of CI; we can simply use artificially inserted missing values to create the counterfactual examples. In other words, to evaluate the effect of one variable or modality on the final prediction, we assume it is missing in a counterfactual scenario. To demonstrate this feature, we used this technique on the predictions of an mPROMISSING model. Fig. 7 shows the results for four example patients with true and false model decisions in the first and second row, respectively. In the first case (Fig. 7a), the model correctly predicts no-remission, and CI shows that the demographic features are the decisive factor by reducing the probability of remission by ~ 0.5. Similar inference can be used to explain the false decisions. For example, PANSS and MINI measures are decisive factors in wrong model predictions in Fig. 7c and 7d, respectively. This extra level of model explainability at the individual patient level is valuable in the clinical settings as it answers the "Why?" question often asked by the clinicians about decisions of black box models, and hopefully, paves the way towards the new field of precision psychiatry (Fernandes et al., 2017).

4 Summary and Conclusions

In this work, we presented PROMISSING, a promising *second* thought on dealing with missing data in machine learning. Unlike previous works, we do not intend to replace missing values based on their relation to other features or the joint distribution of observed data. Instead, we opt to use the unknowns as an additional source of information by learning a representation for missing data. The learned representation is used to prune missing values by neutralizing their effect on the neurons' activation. The proposed method is convenient, intuitive, and reliable. It is convenient because it is straightforward in implementation and application; there is no need for extra data preparation, and the missing values are directly fed to the model; and unlike some data imputation methods, it is flexible with all input data types including continuous, binary, and categorical. It is intuitive because it reacts to unknowns like a living agent; the more the unknowns, the less the actions. Moreover, it is reliable since it performs as well as its alternatives while its decisions are calibrated with the amount of information it receives. Therefore, we consider PROMISSING as one step ahead towards more naturalistic ML-based decision-making in uncertain environments. Despite our extensive experiments, several analytical and empirical aspects of PROMISSING remains unexplored. Exploring the connection between modeling unknowns and model uncertainty, empirical comparison with model-based incomplete data modeling, and applications on more complex and structured data are among the unexplored avenues that we consider as future work.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of conclusions of this work, we have made the core implementation of the proposed method available in appendix A.1. The `nanDense` layer can be directly imported and tested by the readers. Furthermore, anonymized scripts and codes to reproduce the results for experiments on simulated data and OpenML datasets are included in the supplementary materials. Meanwhile, in case needed, we are open to sharing scripts of the experiments on clinical data with reviewers during the discussion process (the data is not available publicly). In our largest validation experiments in Sec. 3.2, we have used 52 publicly available Ml datasets from OpenML. We tried our best to be very specific, descriptive, and precise about the applied parameter settings, model specifications, and model evaluations in Sec. 3. We plan to release all the codes and scripts in Github upon acceptance of the manuscript.

REFERENCES

Nancy C Andreasen, William T Carpenter Jr, John M Kane, Robert A Lasser, Stephen R Marder, and Daniel R Weinberger. Remission in schizophrenia: proposed criteria and rationale for consensus. *American Journal of Psychiatry*, 162(3):441–449, 2005.

Melissa J Azur, Elizabeth A Stuart, Constantine Frangakis, and Philip J Leaf. Multiple imputation by chained equations: what is it and how does it work? *International journal of methods in psychiatric research*, 20(1):40–49, 2011.

Yoshua Bengio and Francois Gingras. Recurrent neural networks for missing or asynchronous data. *Advances in neural information processing systems*, pp. 395–401, 1996.

Cesar F Caiafa, Ziyao Wang, Jordi Sole-Casals, and Qibin Zhao. Learning from incomplete features by simultaneous training of neural networks and sparse coding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2621–2630, 2021.

Francois Chollet et al. Keras, 2015. URL `https://github.com/fchollet/keras`.

Brisa S Fernandes, Leanne M Williams, Johann Steiner, Marion Leboyer, André F Carvalho, and Michael Berk. The new field of precision psychiatry. *BMC medicine*, 15(1):1–7, 2017.

Pedro J García-Laencina, José-Luis Sancho-Gómez, and Aníbal R Figueiras-Vidal. Pattern classification with missing data: a review. *Neural Computing and Applications*, 19(2):263–282, 2010.

Zoubin Ghahramani and Michael I Jordan. Supervised learning from incomplete data via an em approach. In *Advances in neural information processing systems*, pp. 120–127, 1994.

Zoubin Ghahramani and Michael I Jordan. Learning from incomplete data. 1995.

Shih-Cheng Huang, Anuj Pareek, Roham Zamanian, Imon Banerjee, and Matthew P Lungren. Multimodal fusion with deep neural networks for leveraging CT imaging and electronic health record: a case-study in pulmonary embolism detection. *Scientific reports*, 10(1):1–9, 2020.

Sebastian Jäger, Arndt Allhorn, and Felix Bießmann. A benchmark for data imputation methods. *Frontiers in big Data*, pp. 48, 2021.

Julie Josse, Nicolas Prost, Erwan Scornet, and Gaël Varoquaux. On the consistency of supervised learning with missing values. *arXiv preprint arXiv:1902.06931*, 2019.

René S Kahn, Inge Winter van Rossum, Stefan Leucht, Philip McGuire, Shon W Lewis, Marion Leboyer, Celso Arango, Paola Dazzan, Richard Drake, Stephan Heres, et al. Amisulpride and olanzapine followed by open-label treatment with clozapine in first-episode schizophrenia and schizophreniform disorder (optimise): a three-phase switching study. *The Lancet Psychiatry*, 5 (10):797–807, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Marine Le Morvan, Julie Josse, Thomas Moreau, Erwan Scornet, and Gaël Varoquaux. Neumiss networks: differential programming for supervised learning with missing values. In *Advances in Neural Information Processing Systems 33*, 2020.

Xuejun Liao, Hui Li, and Lawrence Carin. Quadratically gated mixture of experts for incomplete data classification. In *Proceedings of the 24th International Conference on Machine learning*, pp. 553–560, 2007.

Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019.

Marine Le Morvan, Julie Josse, Erwan Scornet, and Gaël Varoquaux. What's a good imputation to predict with missing values? *arXiv preprint arXiv:2106.00311*, 2021.

Ramaravind K Mothilal, Amit Sharma, and Chenhao Tan. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pp. 607–617, 2020.

Robert K Nowicki, Rafal Scherer, and Leszek Rutkowski. Novel rough neural network for classification with missing data. In *2016 21st International Conference on Methods and Models in Automation and Robotics (MMAR)*, pp. 820–825. IEEE, 2016.

Zdzislaw Pawlak. Rough set theory and its applications to data analysis. *Cybernetics & Systems*, 29 (7):661–688, 1998.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

Joseph L Schafer and John W Graham. Missing data: our view of the state of the art. *Psychological methods*, 7(2):147, 2002.

Marek Smieja, Łukasz Struski, Jacek Tabor, Bartosz Zieliński, and Przemysław Spurek. Processing of missing data by neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 2724–2734, 2018.

Alex J Smola, SVN Vishwanathan, and Thomas Hofmann. Kernel methods for missing variables. In *International Workshop on Artificial Intelligence and Statistics*, pp. 325–332. PMLR, 2005.

Stef Van Buuren and Karin Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of statistical software*, 45(1):1–67, 2011.

Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

David Williams, Xuejun Liao, Ya Xue, and Lawrence Carin. Incomplete-data classification using logistic regression. In *Proceedings of the 22nd International Conference on Machine learning*, pp. 972–979, 2005.

Jing Xia, Shengyu Zhang, Guolong Cai, Li Li, Qing Pan, Jing Yan, and Gangmin Ning. Adjusted weight voting algorithm for random forests in handling missing values. *Pattern Recognition*, 69: 52–60, 2017.

Jinsung Yoon, James Jordon, and Mihaela Schaar. Gain: Missing data imputation using generative adversarial nets. In *International Conference on Machine Learning*, pp. 5689–5698. PMLR, 2018.

A APPENDIX

A.1 A NANDENSE LAYER

A preliminary implementation of a `nanDense` layer is listed in Listing 1. This layer is implemented by inheriting and overloading the class constructor, `build`, and `call` functions. A flag, `use_c`, is added to the class constructor to decide whether to use compensatory weights or not (defaulted to `False`). Using this flag, the user can simply switch between PROMISSING and mPROMISSING. Furthermore, the flag `use_bias` is fixed to `True`. In the `build` function an extra weight is concatenated to the end of the weight vector to implement compensatory weight in the case `use_c==True`. The matrix of epsilons is also pre-set for usage in the `call` function. In the `call` function, the values for compensatory weights and neutralizers are computed, accordingly. The current implementation is limited to non-sparse 2-dimensional inputs. The implementation is preliminary and not optimized for memory and speed efficiencies. This implementation is tested with `Python 3.8.3`, `TensorFlow 2.4.1`, and `Keras 2.4.3`. A similar guideline may be followed to implement PROMISSING within other types of NN layers.

```
import keras
import tensorflow as tf
from tensorflow.python.framework import sparse_tensor
from tensorflow.python.ops import gen_math_ops
from tensorflow.python.ops import math_ops
from tensorflow.python.ops import nn_ops
from tensorflow.python.keras import backend as K
from tensorflow.python.framework import dtypes
from tensorflow.python.framework import tensor_shape
from tensorflow.python.keras.engine.input_spec import InputSpec
from tensorflow.python.keras import activations
from tensorflow.python.keras import initializers
from tensorflow.python.keras import regularizers
from tensorflow.python.keras import constraints


class nanDense(keras.layers.Dense):

    def __init__(self,
                 units,
                 use_c = False, # A flag to use compensatory weight or
    not.
                 activation=None,
                 kernel_initializer='glorot_uniform',
                 bias_initializer='zeros',
                 kernel_regularizer=None,
                 bias_regularizer=None,
                 activity_regularizer=None,
                 kernel_constraint=None,
                 bias_constraint=None,
                 **kwargs):
        super(nanDense, self).__init__(units,
              activity_regularizer=activity_regularizer, **kwargs)

        self.use_c = use_c
        self.use_bias = True
        self.units = int(units) if not isinstance(units, int) else units
        self.activation = activations.get(activation)
        self.kernel_initializer = initializers.get(kernel_initializer)
        self.bias_initializer = initializers.get(bias_initializer)
        self.kernel_regularizer = regularizers.get(kernel_regularizer)
        self.bias_regularizer = regularizers.get(bias_regularizer)
        self.kernel_constraint = constraints.get(kernel_constraint)
        self.bias_constraint = constraints.get(bias_constraint)
        self.input_spec = InputSpec(min_ndim=2)
        self.supports_masking = True

```

```
    def build(self, input_shape):

        dtype = dtypes.as_dtype(self.dtype or K.floatx())
        if not (dtype.is_floating or dtype.is_complex):
          raise TypeError('Unable to build `nanDense` layer with non-floating point '
                          'dtype %s' % (dtype,))

        input_shape = tensor_shape.TensorShape(input_shape)
        last_dim = tensor_shape.dimension_value(input_shape[-1])
        if last_dim is None:
            raise ValueError('The last dimension of the inputs to `nanDense` '
                             'should be defined. Found `None`.')
        self.input_spec = InputSpec(min_ndim=2, axes={-1: last_dim})

        if self.use_c:      # an extra weight if use_c is True.
            self.kernel = self.add_weight(
                'kernel',
                shape=[last_dim+1, self.units],
                initializer=self.kernel_initializer,
                regularizer=self.kernel_regularizer,
                constraint=self.kernel_constraint,
                dtype=self.dtype,
                trainable=True)
        else:
            self.kernel = self.add_weight(
                'kernel',
                shape=[last_dim, self.units],
                initializer=self.kernel_initializer,
                regularizer=self.kernel_regularizer,
                constraint=self.kernel_constraint,
                dtype=self.dtype,
                trainable=True)

        self.bias = self.add_weight(
            'bias',
            shape=[self.units,],
            initializer=self.bias_initializer,
            regularizer=self.bias_regularizer,
            constraint=self.bias_constraint,
            dtype=self.dtype,
            trainable=True)

        self.epsilon = tf.fill(self.kernel.shape, K.epsilon()) # Epsilon Matrix

        self.built = True


    def call(self, inputs):

        dtype = self._compute_dtype_object

        if self.use_c:  # Computing and concatenating the compensatory weight to the weights
            c = tf.math.reduce_sum(tf.cast(tf.math.is_nan(inputs), dtype), axis=1)
            c = tf.math.divide(c, inputs.shape[1])
            inputs = tf.concat([inputs,tf.expand_dims(c, axis=1)], axis=1)

        kernel = math_ops.add(self.epsilon, self.kernel) # Adding epsilon to weights

```

```
        if self.dtype:
            if inputs.dtype.base_dtype != dtype.base_dtype:
              inputs = math_ops.cast(inputs, dtype=dtype)

        rank = inputs.shape.rank
        if rank == 2 or rank is None:
            if isinstance(inputs, sparse_tensor.SparseTensor):
                raise NotImplementedError
            else:
                outputs = []
                for i in range(self.kernel.shape[1]): # Computing Neutralizers and activations for each neuron
                    d = tf.math.divide(-self.bias[i]/self.kernel.shape[0], kernel[:,i])
                    temp_inputs = tf.where(tf.math.is_nan(inputs), d, inputs) # replacing nans in inputs with -b/w
                    outputs.append(gen_math_ops.mat_mul(temp_inputs, kernel[:,i:i+1]))
                outputs = tf.concat(outputs, axis=1)

        # Broadcast kernel to inputs.
        else:
            raise NotImplementedError

        outputs = nn_ops.bias_add(outputs, self.bias)

        if self.activation is not None:
            outputs = self.activation(outputs)

        return outputs
```

Listing 1: An implementation of a `nanDense` layer in Keras.

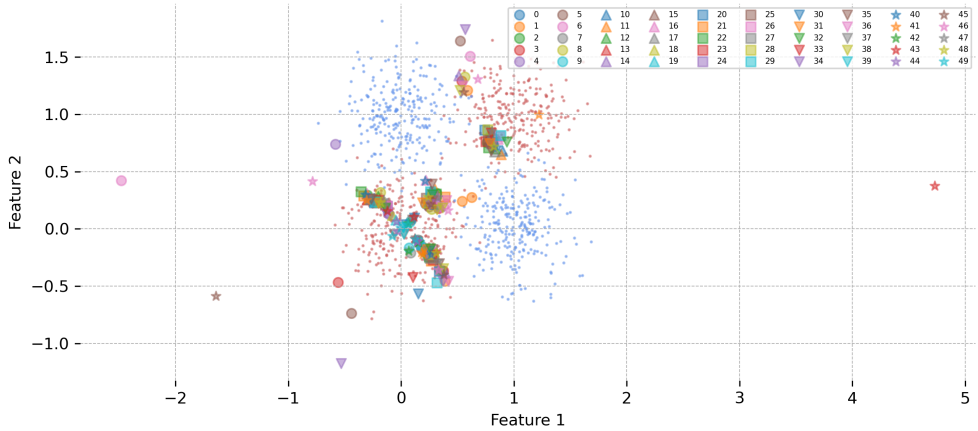


Figure 8: Patterns of neutralizer elements across two input features and the four neurons across 50 simulation repetitions. The markers with the same color and shape show the neutralizer values for each of four neurons in a specific run (the legends) of the pipeline. The blue and red dots in the background are representing the simulated data.

A.2 Supplementary Material for the Simulation Study

Fig. 8 shows the values of $\boldsymbol{U}$ across 50 simulation runs in our simulation study. In our simulation study, $\boldsymbol{U}$ has eight values (two inputs for each four input neurons). Markers with similar color and shape represent the neutralizer values for two features (axes) in four neurons in a specific run. For example, the four blue circles show the values of four neutralizers in two-dimensional feature

space in the first run of the experiment. Since the scatter plot is very crowded, we plotted only the 8 representative runs in Fig. 3 in the main text in Sec. 3.1. Our visual inspection shows that the network finds four main groups of solutions for representing unknowns. See Sec. 3.1 for detailed descriptions about these four groups.

A.3 Supplementary Materials for Experiments on OpenML Data

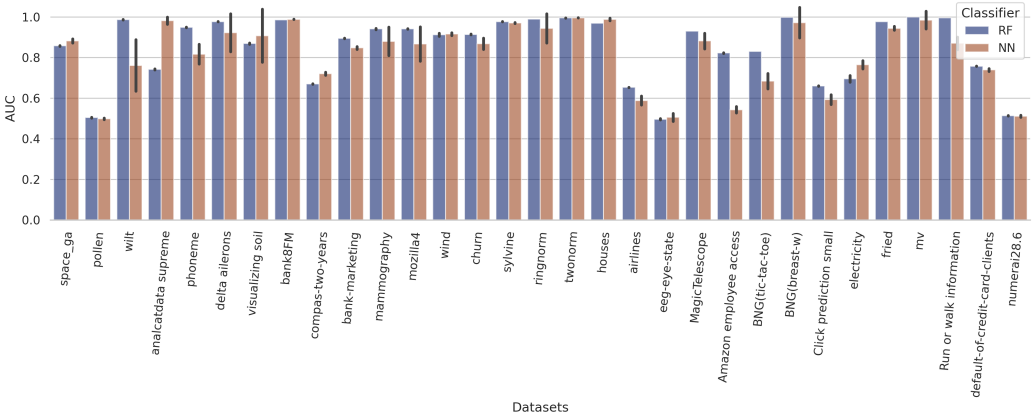
Table 2 and Table 3 summarize the OpenML datasets that are respectively used in our classification and regression analyses in Sec. 3.2. We used exactly the same datasets as in Jäger et al. (2021).

Table 2: Classification datasets that are used in our analysis in Sec. 3.2.

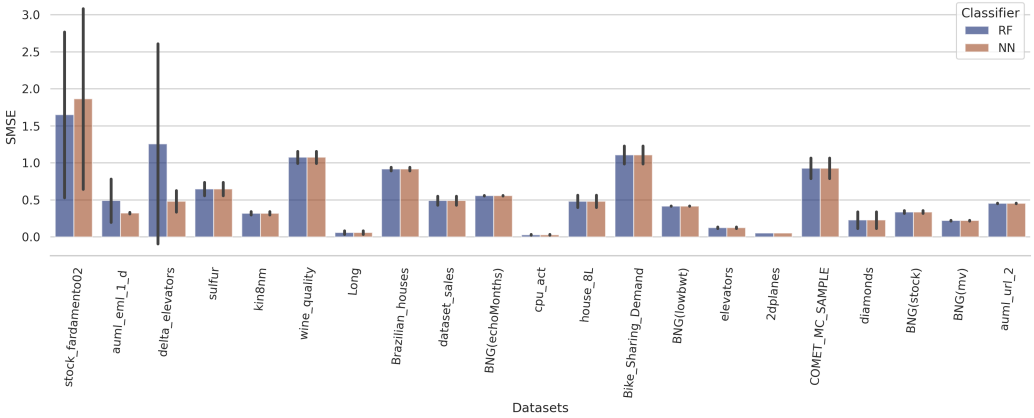
	Dataset Name	OpenML ID	Sample Num.	Feature Num.
1	space-ga	737	3107	6
2	pollen	871	3848	5
3	wilt	40983	4839	5
4	analcatdata-supreme	728	4052	7
5	phoneme	1489	5404	5
6	delta-ailerons	803	7129	5
7	visualizing-soil	923	8641	3
8	bank8FM	725	8192	8
9	compas-two-years	42192	5278	7
10	bank-marketing	1558	4521	7
11	mammography	310	11183	6
12	mozilla4	1046	15545	5
13	wind	847	6574	14
14	churn	40701	5000	16
15	sylvine	41146	5124	20
16	ringnorm	1496	7400	20
17	twonorm	1507	7400	20
18	houses	823	20640	8
19	airlines	42493	26969	2
20	eeg-eye-state	1471	14980	14
21	MagicTelescope	1120	19020	11
22	Amazon-employee-access	4135	32769	0
23	BNG(tic-tac-toe)	137	39366	0
24	BNG(breast-w)	251	39366	9
25	Click-prediction-small	1220	39948	9
26	electricity	151	45312	7
27	fried	901	40768	10
28	mv	881	40768	7
29	Run-or-walk-information	40922	88588	6
30	default-of-credit-card-clients	42477	30000	23
31	numerai28.6	23517	96320	21

Fig. 9a and Fig. 9b compare the performance of our naive NN architecture (that are used in our experiments in Sec. 3.2) with a random forest (RF) classifier or regressor across (with default settings) benchmark datasets. The errorbars represent the variation in AUC/SMSE across 10 repetitions. The models are evaluated in a 2-fold cross-validation scheme. The random number generator seed is fixed for all algorithms in each repetition. Our results show that our simple NN architecture with two hidden layers generally performs as well as random forest models. Both NN and RF models show near the chance performance on 'pollen', 'egg-eye-state', and 'numerai28.6', thus we have removed these datasets from our analyses in Sec. 3.2. About the regression tasks, again we observed poor performance (SMSE$>$1) of both models in the case of 'stock-fardamento02', 'wine-quality', and 'Bike-Sharing-Demand' datasets. Thus, they are removed from our analysis.

Fig. 10 shows the results for experiments in Sec 3.2 in the regression tasks. Plots show the difference between the SMSE of the full model (trained on complete data) and the model trained on data with



(a) AUCs in classification tasks. The higher score is better.



(b) SMSEs in regression tasks. The less score is better.

Figure 9: Comparison between the performance of our simple NN architecture with random forest across a) 31 classification and b) 21 regression tasks.

Table 3: Regression datasets that are used in our analysis in Sec. 3.2.

	Dataset Name	OpenML ID	Sample Num.	Feature Num.
1	stock-fardamento02	42545	6277	6
2	auml-eml-1-d	42675	4585	11
3	delta-elevators	198	9517	7
4	sulfur	23515	10081	7
5	kin8nm	189	8192	9
6	wine-quality	287	6497	12
7	Long	42636	4477	20
8	Brazilian-houses	42688	10692	9
9	dataset-sales	42183	10738	15
10	BNG(echoMonths)	1199	17496	7
11	cpu-act	197	8192	22
12	house-8L	218	22784	9
13	Bike-Sharing-Demand	42712	17379	9
14	BNG(lowbwt)	1193	31104	3
15	elevators	216	16599	19
16	2dplanes	215	40768	11
17	COMET-MC-SAMPLE	23395	89640	6
18	diamonds	42225	53940	7
19	BNG(stock)	1200	59049	10
20	BNG(mv)	1213	78732	8
21	auml-url-2	42669	95911	13

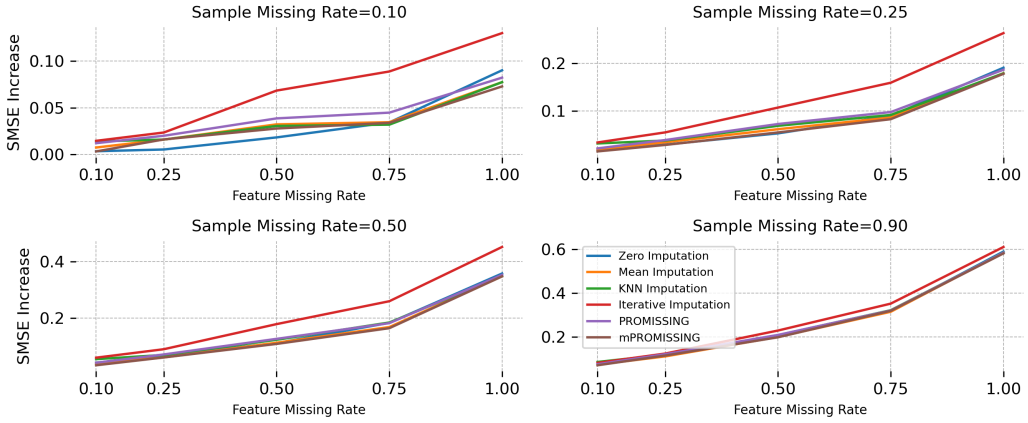


Figure 10: Comparing increase in SMSEs with respect to the full model across four different imputation methods and (m)PROMISSING in regression tasks.

missing values (averaged across 10 repetitions and datasets). As expected the difference in SMSE increases with an increase in the ratios of missing samples and features. The (m)PROMISSING shows competitive performance compared to the better performing imputers, *i.e.*, the zero, mean, and KNN imputers. Especially, mPROMISSING performs slightly better when the ratio of missing features is increasing.