CLASSIFICATION OF INCOMPLETE DATA USING AUGMENTED MLP

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce a new way to train a Multi-Layer Perceptron (MLP) to classify incomplete data. To achieve this, we train an MLP using a two-phased approach. In the first phase, we train an MLP using complete data. Before the second phase of training, we create an augmented dataset. For this, we use non-missing data, delete each feature once, and then fill it using some predefined points. After that, in the second phase, we retrain the network using the augmented dataset. The aim of this type of training is to predict the class label of an incomplete dataset. At the time of testing, when a feature vector with a missing value appears, we initially impute it using the predefined points and find the class label of the feature vector using the trained network. We compare the proposed method with an original MLP on twelve datasets using four imputation strategies. The proposed method's performance is better compared to the originally trained MLP.

1 INTRODUCTION

Handling missing data is a common problem while handling physical world datasets. Let, $\mathbf{X}$ be the data set and $\mathbf{x}_k \in \mathbf{X} \subseteq \mathbb{R}^p$. Now $\mathbf{X}$ is an incomplete dataset if $\mathbf{x}_k$ has $q \in \{1, 2, \cdots, p\}$ missing values. Many real-life systems are affected due to incomplete data García-Laencina et al. (2010). The two main types of missing data are as Little & Rubin (2014): (1) Missing completely at random (MCAR) and (2) Missing at random (MAR). Missing values are handled using various ways. A simple strategy to handle missing values is to delete the data points with missing values. But this procedure is not performed well when a large number of data contains missing values. The second process of handling missing values is filling the missing values using some judiciously chosen techniques. This process is known as imputation. There are various techniques to impute missing values. In García-Laencina et al. (2010), authors clustered it into two groups. The first one is statistical imputation methods, and the second one is imputation based on machine learning-based methods. In the first method, i.e., in Statistical based methods, missing values are imputed by some statistical methods. For example, in zero imputation, incomplete values are imputed by zeros. In the case of random imputation, incomplete values are filled by some random values. In mean imputation, incomplete values are imputed by the mean values of the non-missing data points of the complete features.

When we consider missing value handling using machine learning techniques, $k-$nearest neighbor imputation is a popular machine learning imputation method. If $k = 1$, we say this as $1-$nearest neighbor imputation method. $k-$nearest neighbor ($k-$NN) Dixon (1979) is a popular process where the incomplete value(/s) are imputed by the complete feature value of the nearest neighbors. Here, distances are measured using the observed subspace, and the missing data are imputed by the mean or median value of the corresponding k nearest neighbor data point feature. In $k-$NNI, normally, Euclidean distance is used while the distance is measured. After imputation, there are various methods to check the performance of imputation. In many cases, after that, the performance of imputation is examined using classification or clustering. Here, we trained a network in such a way that it classifies incomplete data.

In Choudhury & Pal (2021b; 2019b; 2021a; 2019a; 2022) the authors modify $1-$NN imputation method and proposed a modified version of $1-$NN imputation for initial imputation of missing values. Here $1-$NN imputation is done using some predefined points. They do not use total training data for $1-$NN imputation. Here, in this article, we also use a similar modified $1-$NN imputation technique to impute the missing values. Similar to Choudhury & Pal (2021b; 2019b; 2021a; 2019a), here, we train a multi-layer perceptron (MLP) using an augmented dataset and we fill it using the modified $1-$NN imputation technique at the testing time. After the initial imputation, we put the complete feature vector into the newly trained MLP and try to find the class label of the incomplete datum.

The remaining part of the article is organized as follows. In the next section i.e., in Section 2 describes the Methodology. Experimental results and analysis are discussed in Section 3. After that, an extension of the proposed method is discussed in Section 4. In the last section i.e., Section 5 concludes the paper.

2 METHODOLOGY

In this section, we describe the methodology for doing the experiments. We use a modified version of Multi-Layer Perceptron (MLP) for computing all operations. We begin by describing the MLP classifier used in our experiment. Then we describe the training procedure of MLP and the testing procedure using the trained MLP.

2.1 MULTI LAYER PERCEPTRON

A multi-layer perceptron (MLP) has three parts: the input layer, hidden layers, and output layer.

Let there be a single hidden layer in our proposed MLP model. In the first layer, i.e. in the input layer, p number of input features are present. In the second layer i.e., in the hidden layer q nodes are present, and the activation function of each node is $\delta(\cdot)$. In the last layer, i.e., in the output layer, the target output of j^{th} node of k^{th} data point is t_{kj}, the total data points are n, and the total number of output nodes i.e, number of classes for a particular problem is r. There are different types of MLP present in the literature. From that, we adopt the MLP which is described in Kumar (2004). In MLP, the first layer, i.e. the input layer is a fan-out layer. That means the input and output of the first layer are the same. Mathematically, it is like this:

$$\begin{aligned} \delta\left(x_{ki}\right) = x_{ki}, i = 1, \cdots, p; \\ \delta\left(x_{k0}\right) = 1, \forall k; \end{aligned} \tag{1}$$

here the i^{th} feature of the input vector $\mathbf{x}_k$ is x_{ki} and the bias of input layer is $\delta\left(x_{k0}\right)$. For the second layer, i.e., in the hidden layer:

$$\begin{aligned} z_{kh} = \sum_{i=0}^{p} w_{ih}\delta\left(x_{ki}\right), h = 1, \cdots, q \\ \delta\left(z_{kh}\right) = max(0, z_{kh}), h = 1, \cdots, q; \\ \delta\left(z_{k0}\right) = 1, \forall k; \end{aligned} \tag{2}$$

where the connection between h^{th} hidden node and i^{th} input node is w_{ih}, the bias connection of the h^{th} hidden neuron is w_{0h} and the bias of hidden layer is $\delta\left(z_0\right)$. For the last layer, i.e. the output layer:

$$\begin{aligned} y_{kj} = \sum_{h=0}^{q} w_{hj}\delta\left(z_{kh}\right), j = 1, \cdots, r \\ o_{kj} = \delta\left(y_{kj}\right) = max(0, y_{kj}), j = 1, \cdots, r. \end{aligned} \tag{3}$$

here the weight between j^{th} output node and h^{th} hidden node is w_{hj} and the bias of the j^{th} output node is w_{0j}. So, the instantaneous system error for the k^{th} training pattern can be written as follows:

$$E^k = -\sum_{j=1}^{r} t_{kj} log(o_{kj}) \tag{4}$$

For a classifier, typically, $t_{kj} \in \{0, 1\}$. Instead of cross-entropy, one can use other loss functions such as square loss. Similarly, in place of the rectified linear unit (ReLU) activation function, other choices can also be used like the sigmoidal function. The network connection weights are randomly initialized and then they are updated to reduce the loss. There are different learning methods including the back-propagation method Kumar (2004).

Algorithm 1 : Training of the MLP using augmented dataset

```
INPUT: $X_{TR}$: Training data set; $p$: Number of features; $r$: Number of classes; $n_c, c = 1, 2, \cdots, r$: Number of clusters in class $c$; $N_1$: Number of epochs for the first phase of training; $N_2$: Number of epochs for the second phase of training.
BEGIN
1: Set $\widetilde{V} = \emptyset$, $\hat{V} = \emptyset$.
2: while i=1 to $r$ do
3:     Set $\hat{X}_i = \{\mathbf{x}|\mathbf{x} \in X_{TR}$ and $\mathbf{x}$ belongs to the $i^{th}$ class$\}$.
4:     Set $\widetilde{\mathbf{v}}_i = \frac{1}{|\hat{X}_i|} \sum_{\mathbf{x}\in\hat{X}_i} \mathbf{x}$.
5:     $\widetilde{V} = \widetilde{V} \cup \widetilde{\mathbf{v}}_i$.
6:     Divide $\hat{X}_i$ using the $k$-means algorithm into $n_c$ clusters and extract $n_c$ cluster centers $\hat{V}_i = \{\hat{\mathbf{v}}_{i1}, \hat{\mathbf{v}}_{i2}, \cdots, \hat{\mathbf{v}}_{in_c}\}$ ; $\hat{\mathbf{v}}_i \in \mathbb{R}^p$.
7:     $\hat{V} = \hat{V} \cup \hat{V}_i$.
8: end while
9: Set $\widetilde{V'} = \widetilde{V} \cup \hat{V}$.
10: while s=1 to $p$ do
11:     To create $X^s$, in every $\mathbf{x} \in X_{TR}$, we find out $\mathbf{x}^s$ by replacing $x_s$ (the $s^{th}$ component of $\mathbf{x}$) using $v_{ls}$ if $l = argmin_{k'} \left\{||\mathbf{x} - \widetilde{\mathbf{v}'}_{k'}||_*^2\right\}$, where $||.||_*$ is measured using all features except $s^{th}$ feature.
12: end while
13: Trained an MLP by $X_{TR}$ for $N_1$ epochs.
14: Retrained the same MLP for $N_2$ epochs by $X^{Total} = X_{TR} \cup \{X^1 \cup X^2 \cup \cdots \cup X^p\}$. For all $\mathbf{x} \in X_{TR}$ and its corresponding $\mathbf{x}^s \in X^s$, the target vector is same as the class label of $\mathbf{x} \in X_{TR}$.
END
```

2.2 TRAINING PROCEDURE

Let, $X \subseteq R^p$ be a dataset. Now each data point of $X \subseteq R^p$, $\mathbf{x}_i \in R^p$ has a class label c_i, where $c_i \in \{1, \cdots, r\}$. For training and testing we divide $X \subseteq R^p$ into two equal parts $X = X_{TR} \cup X_{TE}, X_{TR} \cap X_{TE} = \phi$. From these two parts, we use X_{TR} for training and X_{TE} for testing.

In the proposed method we assume that $X = X_{TR} \cup X_{TE}, X_{TR} \cap X_{TE} = \phi$ in such a way that all r classes data points are present in X_{TR}. Now let, $X_{TR} = \cup_{i=1}^{r} \hat{X}_i$, where $\hat{X}_i = \{\mathbf{x}_k | \mathbf{x}_k \in$ the i^{th} class$\}$.

Before training, we try to find out some predefined points from X_{TR}. First, we find out some predefined points using the class center. So, the i^{th} *class* center is as:

$$\widetilde{\mathbf{v}}_i = \frac{1}{n_i} \sum_{\mathbf{x}_j \in \hat{X}_i} \mathbf{x}_j. \tag{5}$$

As there are r classes, we find out r predefined points as: $\widetilde{V} = \{\widetilde{\mathbf{v}}_1, \widetilde{\mathbf{v}}_2, \cdots, \widetilde{\mathbf{v}}_r\}$; $\widetilde{\mathbf{v}}_i \in R^p$. After finding the predefined points from cluster centers, we try to increase the predefined points by clustering $\hat{X}_i$ into n_c clusters. Here, n_c is the user-defined cluster number. For clustering, we can use any clustering algorithm. The $k-$means algorithm is used as the clustering algorithm.

As there are r classes, in that step we find $(n_c \times r)$ cluster centers as predefined points: $\hat{V} = \left\{\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \cdots, \hat{\mathbf{v}}_{(n_c \times r)}\right\}$; $\hat{\mathbf{v}}_i \in R^p$.

Then we combine first set of predefined points with second set of predefined points as: $\widetilde{V'} = \hat{V} \cup \widetilde{V}$. Thus total $(r + n_c \times r)$ predefined points are generated. After, finding predefined points we try to create an augmented dataset from the training dataset X_{TR}. The augmented dataset creation procedure is as follows. For creating an augmented dataset, we delete each feature of X_{TR} and impute it using the predefined points. The augmented dataset is a collection of p modified data sets. The procedure to generate p new data set is $X^s, s = 1, 2, \cdots, p$ as follows. In X^s, for $\mathbf{x}_j \in X_{TR}$, x_{js} is modified by v_{ls} if $l = argmin_{k'} \left\{||\mathbf{x}_j - \widetilde{\mathbf{v}'}_{k'}||_*^2\right\}$.

After creating an augmented dataset, we train MLP in two phases. In the first phase, we train the MLP using X_{TR} for N_1 epochs. Then in the second phase, we retrain the same network for N_2 epochs using $X^{Total} = X_{TR} \cup_{s=1}^{p} X^s$. For any $\mathbf{x}_j \in X_{TR}$ and $\mathbf{x}_j^s \in X^s$, the target vector is taken as the class label of $\mathbf{x}_j \in X_{TR}$.

The whole training procedure is summarized in an algorithmic form in Algorithm 1. A similar training procedure to training an auto-encoder is very effective to handle missing values Choudhury & Pal (2021b; 2019b; 2021a; 2019a).

Table 1: Details of the 7 UCI data sets

Dataset	# Instances	# Feature	# Class
Iris	150	4	3
Glass	214	9	7
Seeds	210	7	3
Balance	625	4	3
Monk1	556	6	2
Monk2	601	6	2
Monk3	554	6	2
Vehicle	94	18	4
Pima	768	8	2
Wine	178	13	3
Wdbc	569	32	2
WPBC	194	34	2

2.3 TESTING PROCEDURE

As stated earlier, here, X_{TR} is used for training and X_{TE} is used for testing. We assume that missing values are present in only X_{TE} and each testing data point may have up to $(\frac{P}{2})$ features missing. We impute the missing values of X_{TE} using the predefined points. The s^{th} missing feature value of $\mathbf{x}_i$, x_{is} is filled by v_{ls} if $l = argmin_j \left\{ ||\mathbf{x}_i - \widetilde{\mathbf{v}'}_j||_*^2 \right\}$ where $||.||_*$ is measured only using the observed feature space of $\mathbf{x}_i$. After imputation, we pass $\mathbf{x}_i$ through the trained MLP and generate its class labels.

3 EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we describe the dataset, specifications for training the MLP model, and different results using the MLP.

3.1 DATA SETS

To test the effectiveness of the proposed method, we take 12 well-known data sets Mertz & Murphy (2005). We randomly shuffle it and divide the dataset into two parts: training and testing. In the training dataset, no feature values are missing, whereas in the testing dataset at each point, we randomly remove up to $(\frac{p}{2})$ features. For creating a missing dataset we follow the same procedure as described in Datta et al. (2016). The summary of the 12 datasets used is listed in Table 1.

3.2 EXPERIMENTAL SET UP

In this section, the specifications used to train our MLP model have been given. Here, we describe the number of epochs (N_1 and N_2), nodes in the hidden layer (q), the learning rate to train the network, and the number of clusters (n_c). All results reported here are the average of 25 random runs.

3.2.1 EPOCHS:

We use $N_1 = 1000$ and $N_2 = 1000$ in all cases in our experiment, where N_1 refers to the number of epochs used for training the MLP on a normal dataset and N_2 refers to the number of epochs for training MLP on an augmented dataset. The proposed method uses $N_1 + N_2$ epochs to train the model.

3.2.2 HIDDEN LAYERS:

We use 1 hidden layer in our MLP model for all cases (normal training & augmented training). However, the nodes in the hidden layer vary as per the input features. If there are p input features, we set the number of nodes in the hidden layer q is $10p$, i.e. $q = 10p$.

3.2.3 Learning rate:

Using a few trial and error experiments we set learning rates (LR) = 0.001 for all datasets.

3.2.4 Clusters:

We set the number of clusters (n_c) as described in Algorithm 1 as 2 to reduce the time complexity as much as possible.

3.3 Results and Analysis

In this section, we compare the performance of different imputation methods using an MLP (where we use only N_1 epochs for training) and augmented MLP (where we use both N_1 and N_2 epochs for training).

In Fig. 1, we compare the performance of different imputation methods using an MLP and the act of MLP on a dataset where no missing values are present. Here, we use 12 datasets. If we consider the imputation methods, mean imputation performs the best in 9 datasets among 12 datasets.

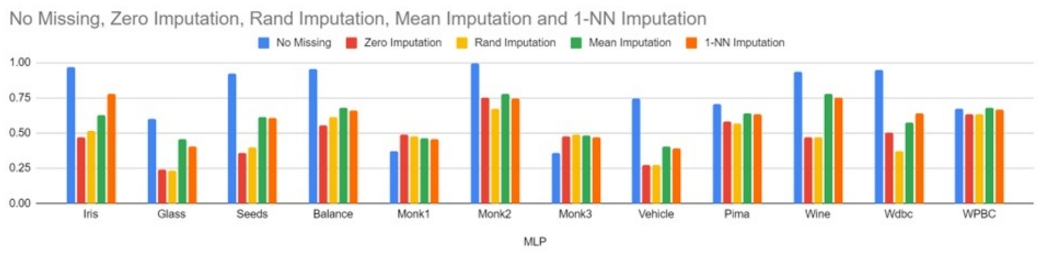

Figure 1: Comparing different imputation method using MLP on different dataset

In Fig. 2, we compare the performance of different imputation methods using augmented MLP (AMLP) and the performance of AMLP where no missing values are present on the same set of 12 datasets. If we consider the imputation methods, mean imputation performs the best in 6 out of 12 datasets, and $1-$NN imputation performs the best on 5 of them.

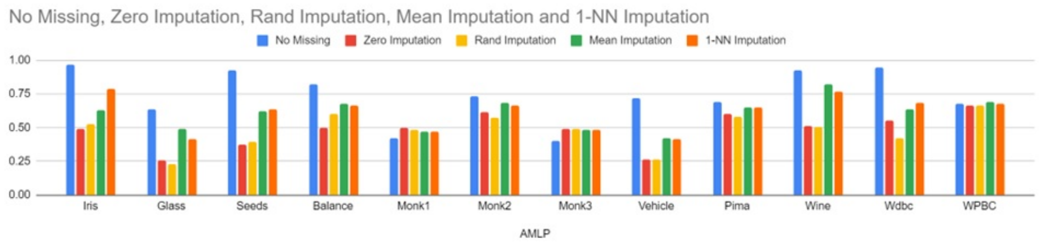

Figure 2: Comparing different imputation method using AMLP on different dataset

Next, we compare the performance of different methods using MLP and AMLP. First, in Fig. 3, we compare the performance of MLP and AMLP where no missing value is present. From Fig. 3, we can conclude that if no missing values are present in a dataset, MLP performs better than AMLP. Among the 12 datasets, MLP classifies better than AMLP in 9 of them.

In Fig. 4, we compare the performance of MLP and AMLP on different datasets when missing values are present, and imputed with zero. Out of 12 cases, AMLP wins in 9 cases. So, we can say that if missing values are present and they are imputed with zero, AMLP performs better than MLP.

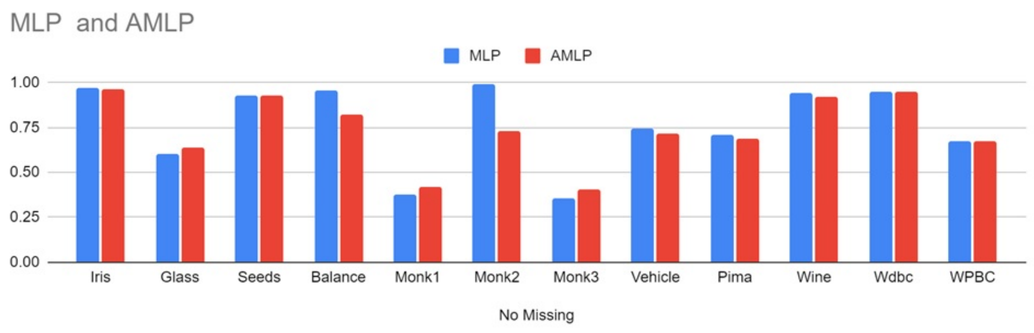

Figure 3: Comparing MLP and AMLP on different dataset when no value is missing

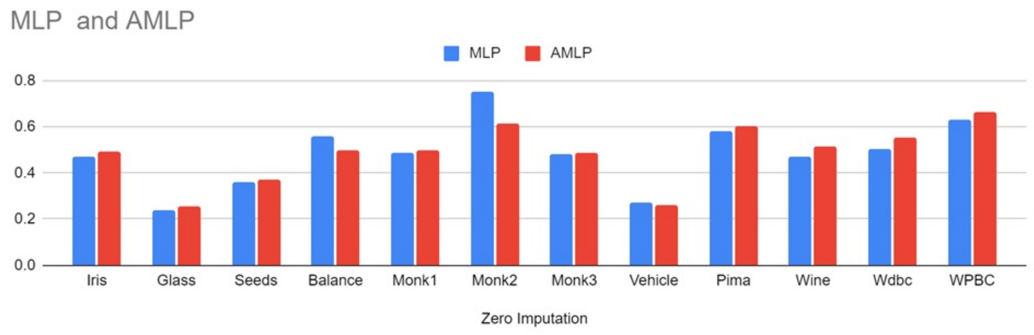

Figure 4: Comparing MLP and AMLP on different dataset when missing values are imputed by zero

In the succeeding figure, i.e., in Fig. 5, we also compare the performance of MLP and AMLP when missing values are present in a dataset and imputed randomly. Here also, AMLP performs better than an MLP. Particularly, out of 12 datasets, AMLP performs better in 7 of them.

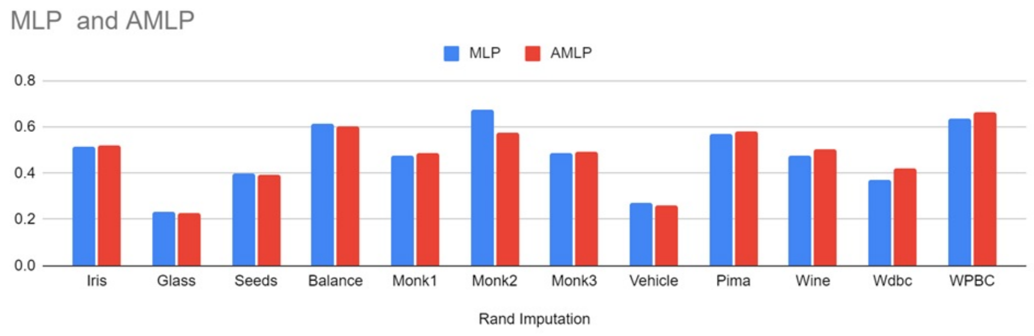

Figure 5: Comparing MLP and AMLP on different dataset when missing values are imputed randomly

In Fig. 6 and Fig. 7, we compare the performance of MLP and AMLP when missing values are present in a dataset using two other imputation strategies. In Fig. 6, the missing values are filled using mean imputation, and in Fig. 7, the missing values are filled using $1-$NN imputation (from some predefined points). In both cases, AMLP performs better than MLP. Particularly, in the first case (in Fig. 6) AMLP performs better in 10 and in the second case (in Fig. 7) AMLP performs better in 11 out of 12 datasets. In conclusion, we can say that if some values are missing in a dataset and it is filled by any method, AMLP performs better than MLP.

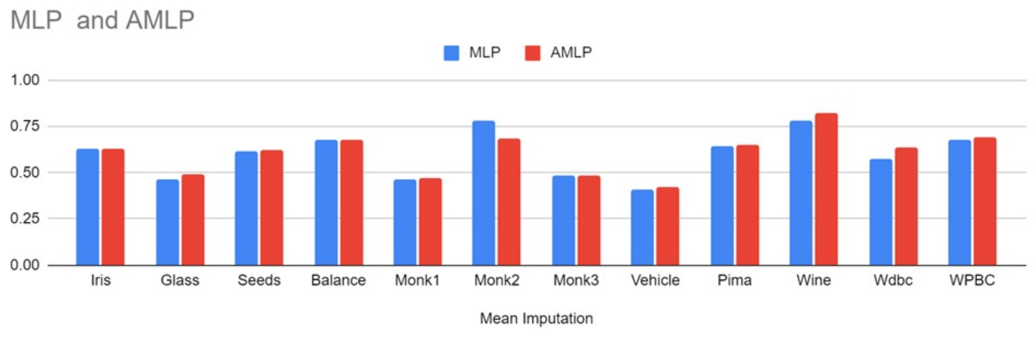

Figure 6: Comparing MLP and AMLP on different dataset when missing values are imputed by mean imputation

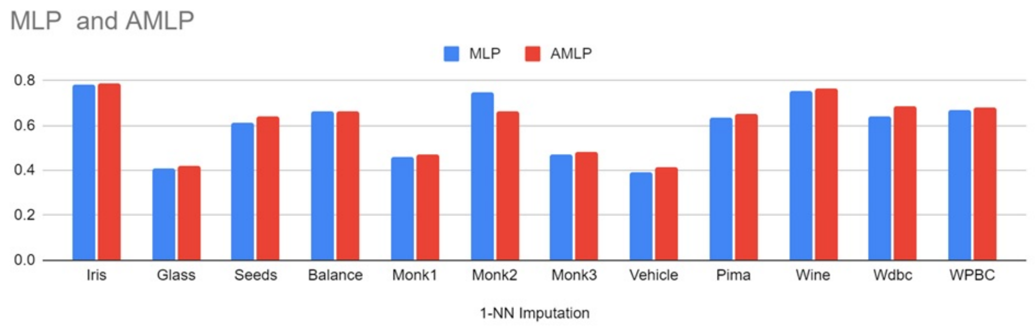

Figure 7: Comparing MLP and AMLP on different dataset when missing values are imputed by 1-NN imputation

3.4 Parameter Sensitivity:

In Table 2 we compare the accuracy of AMLP on 4 datasets (Iris, Monk2, Balance, WPBC) using various n_c (i.e. the number of clusters for identifying predefined points). For comparing the performance we take $n_c = 2, 3, 5$. Similar to previous experiments average of 25 randomly runs is reported here. The detailed result of the experiments is as follows.

In the Iris dataset when we increase n_c the performance varies a little bit; whereas in the Balance dataset when n_c increases the accuracy decreases. The behavior of Monk2 is the opposite of Balance. In Monk2 when n_c increases the accuracy also increases. The behavior of WPBC is a little bit different from the previously described datasets. Here, when n_c increases, first the performance decreases, and then it increases. So, each dataset has an optimal n_c. We can find out an optimal n_c for a dataset using cross-validation.

4 Extended Method

In this section, an extension of the proposed method is discussed. The aim of the proposed method is to train an MLP in such a way that it classifies incomplete data. For that, during training, we use an incomplete datum as input and the original datum class label as the target. In the extended proposed method, with the cross-entropy loss, we try to minimize the latent space representation of the same class data points and, try to maximize the latent space representation of different class label data points. For latent space representation, we use the hidden layer representation of the MLP.

Table 2: Accuracy of AMLP on 4 datasets using various n_c

	Dataset	n_c = 2	n_c = 3	n_c = 5
No Missing	Iris	0.97	0.96	0.97
	Balance	0.82	0.81	0.78
	Monk2	0.73	0.80	0.84
	WPBC	0.68	0.55	0.70
Zero Imputation	Iris	0.49	0.49	0.51
	Balance	0.82	0.81	0.78
	Monk2	0.61	0.63	0.65
	WPBC	0.66	0.63	0.70
Rand Imputation	Iris	0.52	0.53	0.52
	Balance	0.60	0.60	0.60
	Monk2	0.58	0.59	0.61
	WPBC	0.66	0.63	0.70
Mean Imputation	Iris	0.63	0.62	0.62
	Balance	0.68	0.68	0.66
	Monk2	0.68	0.71	0.72
	WPBC	0.69	0.55	0.70
1-NN Imputation	Iris	0.79	0.79	0.78
	Balance	0.66	0.65	0.63
	Monk2	0.66	0.67	0.68
	WPBC	0.68	0.55	0.70

The instantaneous system error for the k^{th} training pattern of the extended proposed method is as follows:

$$E^k = -\sum_{j=1}^{r} t_{kj} log(o_{kj}) + \alpha \sum_{i1=1}^{n} \sum_{i2=1, c_{i1}=c_{i2}}^{n} \left(||\delta\left(Z_{i1}\right) - \delta\left(Z_{i2}\right)||\right) \tag{6}$$

$$- \beta \sum_{i1=1}^{n} \sum_{i2=1, c_{i1} \neq c_{i2}}^{n} \left(||\delta\left(Z_{i1}\right) - \delta\left(Z_{i2}\right)||\right) \tag{7}$$

where, α, β is the coefficient of two regulizer and $\delta\left(Z_i\right)$ is the hidden layer representation of i^{th} data point. We take $\alpha = \beta = 1$ for this experiment.

Here, we proposed two extended methods: MLP with regularizer (MLP+R) and AMLP with regularizer (AMLP+R). We repeat the whole process 5 times with $N_1 = N_2 = 15$ and with LR=0.01. Here, in Table 3, we compare the accuracy of AMLP and MLP with their regularized methods on 4 datasets (Iris, Glass, Seeds, Balance). In most cases, the performance of a method with an augmented dataset is better than the method with the original dataset. But, using a regularizer, there is no effect on the performance. Maybe this is because we use a less number of iterations here. Since at the training phase, in each epoch we have to do $|X_{TR}|^2$ number of calculations, the total computational complexity of the extended proposed method is very high, and we use less number of iterations.

5 Conclusion

Here, we proposed a new way to train an MLP which can handle missing data properly. We compare the proposed method with an MLP on twelve different datasets using four types of imputation methods. In most cases, when we use a newly proposed MLP, the performance improves for missing data handling. The performance of the proposed MLP varies with n_c (i.e., the number of clusters for identifying predefined points). We also extend our method using two regularizers. This type of training method for handling incomplete data can applies to other classifiers also (like the k-NN classifier). In the future, we want to explore this.

Table 3: Accuracy of MLP and AMLP with the regularizer

	Dataset	MLP + R	AMLP+ R	MLP	AMLP
No Missing	Iris	0.69	0.83	0.74	0.92
	Glass	0.32	0.37	0.32	0.39
	Seeds	0.62	0.65	0.46	0.72
	Balance	0.82	0.87	0.84	0.86
Zero Imputation	Iris	0.45	0.45	0.46	0.47
	Glass	0.26	0.26	0.26	0.27
	Seeds	0.40	0.40	0.35	0.38
	Balance	0.56	0.57	0.57	0.56
Rand Imputation	Iris	0.48	0.50	0.49	0.50
	Glass	0.26	0.24	0.26	0.27
	Seeds	0.38	0.42	0.38	0.42
	Balance	0.61	0.62	0.62	0.62
Mean Imputation	Iris	0.58	0.68	0.62	0.68
	Glass	0.31	0.34	0.32	0.36
	Seeds	0.45	0.52	0.39	0.54
	Balance	0.67	0.69	0.69	0.69
1-NN Imputation	Iris	0.63	0.70	0.62	0.73
	Glass	0.31	0.34	0.32	0.36
	Seeds	0.49	0.51	0.42	0.52
	Balance	0.66	0.66	0.65	0.66

REFERENCES

Suvra Jyoti Choudhury and Nikhil R Pal. Classification of incomplete data using autoencoder and evidential reasoning. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pp. 167–177. Springer, 2019a.

Suvra Jyoti Choudhury and Nikhil R Pal. Imputation of missing data with neural networks for classification. *Knowledge-Based Systems*, 2019b. URL `https://doi.org/10.1016/j.knosys.2019.07.009`.

Suvra Jyoti Choudhury and Nikhil R. Pal. Classification of incomplete data integrating neural networks and evidential reasoning. *Neural Computing and Applications*, pp. 1–15, 2021a. doi: 10.1007/s00521-021-06267-1. URL `http://doi.org/10.1007/s00521-021-06267-1`.

Suvra Jyoti Choudhury and Nikhil R Pal. Fuzzy clustering of single-view incomplete data using a multi-view framework. *IEEE Transactions on Fuzzy Systems*, 2022.

Suvra Jyoti Choudhury and Nikhil Ranjan Pal. Deep and structure-preserving autoencoders for clustering data with missing information. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(4):639 – 650, 2021b. URL `https://doi.org/10.1109/TETCI.2019.2949264`.

Shounak Datta, Supritam Bhattacharjee, and Swagatam Das. Clustering with missing features: A penalized dissimilarity measure based approach. *arXiv preprint arXiv:1604.06602*, 2016.

John K Dixon. Pattern recognition with partly missing data. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(10):617–621, 1979.

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.

Satish Kumar. *Neural networks: a classroom approach*. Tata McGraw-Hill Education, 2004.

Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*. John Wiley & Sons, 2014.

J Mertz and PM Murphy. University of california at irvine (uci) repository of machine learning databases. *Available ftp:/ftp. ics. uci. edu/pub/machine-learning-databases*, 2005.