Deep and Structure-Preserving Autoencoders for Clustering Data With Missing Information

Suvra Jyoti Choudhury, *Student Member, IEEE*, and Nikhil Ranjan Pal, *Fellow, IEEE*

***Abstract*—Most real-life data suffer from missing values. Here we deal with the problem of exploratory analysis, via clustering, of data with missing values. For this we need an effective mechanism to deal with missing features so that all available information can be used for clustering. We propose two autoencoder-based methods for handling of missing data for clustering. The autoencoder is trained in a two-phase scheme using only part of the given data set which does not have any incomplete instances in such a manner that the autoencoder is better equipped to deal with incomplete data. To cluster the entire data set which has instances with missing values, we generate the latent space representation of the all instances, with or without, missing information. Before the incomplete instances are submitted to the autoencoder, the missing inputs are filled in by a $k-$nearest neighbor-based rule. The clustering is then done in the latent space using the fuzzy-c-means (FCM) algorithm. In the second method, to preserve the "structure" of the input data in the latent space we extend our method by adding Sammon's stress as a regularizer to the objective function of the autoencoder. We test the effectiveness of the proposed algorithms on several data sets and compare the results with five state-of-the-art techniques. For comparison, we use two performance indicators: Normalized Mutual Information (NMI) and Adjusted Rand index (ARI).**

***Index Terms*—Clustering, fuzzy-c-means, latent space representation, missing data, neural networks, deep network, Sammon's stress.**

I. Introduction

MISSING data is a common issue while handling real world problems. According to [1] any datum with some (but not all) missing feature values is referred to as an incomplete datum and a data set with at least one incomplete datum is referred to as an incomplete data set. Let, $\mathbf{x}_k$ be the k^{th} data point or datum with p features, $\mathbf{x}_k \in \mathbf{X} \subseteq R^p$, $\mathbf{X}$ is the data set. So, if $\mathbf{x}_k$, for some k has q, $0 < q < p$, missing values then $\mathbf{X}$ is called an incomplete dataset. Many real life systems suffer due to missing data [2]. For example, (1) control based applications (traffic monitoring [3], industrial processes [4], management of telecommunications, and computer networks [5]), (2) wireless sensor networks [6], [7], (3) automatic speech recognition [8], [9], (4) financial and business applications [10], [11], and (5) medical diagnosis [12]–[15] suffer due to missing data.

In the literature, in most clustering techniques the missing values are replaced by some values and then a standard clustering technique is applied [1], [16]–[28]. In some cases, new objective functions are formed and clustering is done on the data with missing information; while in other cases, the structure of the incomplete data is captured [29] during clustering.

Deep neural networks, particularly deep autoencoders have many applications including image retrieval, speech recognition, character recognition, and reinforcement learning [30]–[34]. Recently in [35], the authors used an improved variational autoencoder network (VAN) for clustering of data with missing values. To improve the performance they add convolutional kernels in the encoder. They process three image data sets. The clustering is done using the latent space representation with a higher order fuzzy c-means clustering algorithm using tensor distance, which is claimed to capture the correlation in the high-order tensor space.

Here, we propose a deep autoencoder based clustering technique to cluster data with missing completely at random (MCAR) type of missing values. When a data point has missing features we impute it by our proposed method and then project the imputed data and the complete data into a latent space using a deep autoencoder. Then clustering is done in the latent space. We test the effectiveness of the proposed algorithm on twelve challenging datasets and compare the results of the proposed algorithm with five state-of-the-art techniques with respect to NMI [36] and ARI [37]. Our results reveal that the performance of the proposed method is better compared to other methods. We note here that the work in [35] also uses clustering in the latent space produced by a variational autoencoder. We want make a few remarks on this: (1) The work in [35][1] uses a modified variational autoencoder, while we use a conventional autoencoder. (2) [35] minimizes three types of losses (reconstruction, regularization, and clustering), but in the proposed method we minimize only the reconstruction loss; (3) In [35] convolutional layers are used which make it suitable for images and it naturally uses image data sets, while our method is a general one and is not adapted for images - we do not use any image data. (4) while training we use an augmented data set by deleting each feature once (considering only the instances without any missing values) and impute it using a judiciously chosen method. (5) For final clustering, before submitting to the autoencoder, instances with missing values are made complete using a $k-$nearest neighbor

Manuscript received January 17, 2019; revised May 21, 2019 and August 31, 2019; accepted September 29, 2019. Date of publication November 22, 2019; date of current version July 22, 2021. *(Corresponding author: Nikhil Ranjan Pal.)*

The authors are with the Electronics and Communication Sciences Unit, Indian Statistical Institute, Kolkata 700108, India (e-mail: cshuvrajyoti@gmail.com; nikhil@isical.ac.in).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/TETCI.2019.2949264

[1]We discovered this paper while revising as it was submitted and published after we submitted our manuscript.

rule. (6) In [35] the authors use a bottleneck layer as the latent space representation, but we do not use a bottleneck layer. (7) We use the conventional FCM algorithm for clustering, but in [35] authors use a higher-order FCM for clustering.

Almost all authors used autoencoders with a bottleneck layer for handling missing data and for classification problem. Use of a bottleneck layer is useful for feature extraction and data compression. In our view, for handling missing values keeping in view the problem of clustering, the objective should be a bit different from data compression or usual feature extraction. Here we should try to get be a better representation of the inputs in the latent space. Consequently, we use a latent space of dimension higher than that of the input. To make an effective use of the training data that are complete, we use a two-stage training scheme. This training scheme equips the network to deal missing data in a more effective manner. While training, since a gradient descent algorithm is used to minimize the objective function, a good initial choice of the missing value is important. For this, we use a simple but judiciously chosen value. Such a two-stage training scheme has been found to be very effective for classification [38]. The objective of clustering an incomplete data should be to produce the same or almost similar partition that would have been produced if there was no missing data. This may be facilitated if we can preserve the geometry of the input data in the latent space representation and this will help to improve the performance of any distance based clustering algorithm. Keeping this in mind, we augment our method by adding Sammon's stress [39] as a regularizer to the objective function of the autoencoder.

The rest of the article is organized as follows. Sections II and III, respectively, describe the literature and the proposed method. Experimental results are discussed in Section IV. Section V concludes the paper.

II. Survey of Literature

Little and Rubin [40] characterized missing data into three types. They are (1) missing completely at random (MCAR) which means missing of a feature value is independent of the feature itself or any external influence, (2) missing at random (MAR) where although the missing values do not depend on the associated missing variables, the "missingness" can be predicted by other variables, (3) not missing at random (NMAR) where a missing datum is not random and depends on an associated variable. Most of the missing value prediction methods are for MCAR and MAR.

To handle MCAR and MAR type of data different types of analyses are done. The simplest one is complete data analysis [40], [41]. Here the analysis is done only on the complete data, i.e., all incomplete data points are discarded. But this procedure is useful only when a small amount of information is missing.

Another family of methods for handling missing data consists of imputing missing values and then the analysis is done on that imputed data. In [2] the authors state two major types of imputation techniques. These are statistical imputation methods and imputation based on machine learning techniques. In a statistical imputation method, the missing information is imputed by common statistical techniques [40]–[42]. For example, missing data are replaced by the average of complete data points of that feature and this is known as mean imputation (MI); sometimes even the missing information is replaced by zeros, which is known as zero imputation (ZI). Regression imputation obtains a regression model with the complete data points and then the regression model is used to predict the missing data. Of the p features if k features are missing, then we need $k-$regression models assuming at most one missing value per instance. In the cold and hot deck imputation [43], a missing value is imputed using the feature value of the closest complete data point according to the existing features from the same (and sometimes from different) data sources. In the case of multiple imputations, a missing value is imputed by a set of plausible ones that represent the uncertainty about the right value to impute. Thus, the missing values are imputed multiple times and multiple datasets are created. Then the multiple datasets are analyzed by using standard procedures.

There are several other machine learning procedures to impute missing data. For example, $k-$nearest neighbor ($k-$NN) [44] is a common procedure to fill missing information using the nearest neighbor rule where the distance used to find neighbors is computed using the observed subspace (kNNI). In [45] Richard *et al*. modify the traditional $k-$NN rule and proposed a distance-weighted $k-$nearest neighbor rule to classify incomplete data. Troyanskata *et al*. [44] proposed a weighted k-NN imputation (kNNI) and singular value decomposition imputation (SVDI) using $k-$most significant eigenvectors. The SVDI method performs regression-based estimation of the missing data using the k most significant eigenvectors of the dataset. The self-organizing map (SOM) [46], [47] and multi-layer perceptron (MLP) [48] are the other two machine learning techniques to impute missing data. Various other machine learning techniques to impute and classify incomplete data can be found in [49]–[53].

Recently in [54] the authors proposed the Order-Sensitive Imputation for a "clustered missing values" framework, in which missing values are imputed sequentially. The idea is that the values filled earlier in the process are used for subsequent imputation of other missing values. For this process, the authors formulate an optimization problem to find the optimal order of imputation and propose two heuristic algorithms to solve it. In [55] first, for each class, a prototype is formed. The imputation of missing values is then done using all prototypes. Thus, for each incomplete instance, r complete instances are generated, when the total number of classes is r. Now, each new data point is classified using any well-known classifier. Then using a prototype-based credal classification (PCC) method, the results of the classification are combined to find the appropriate class label of the incomplete instance. Sometimes, incomplete patterns that are difficult to classify will be reasonably and automatically committed to some meta-class by the PCC method. Sometimes, missing entries of a matrix are restored using a matrix completion method. For example, in [56] locally linear reconstruction is used to find missing entries of a matrix. Here, the authors propose a new algorithm called locally linear approximation (LLA) which tries to keep the local structure of the data space while restoring the missing entries. The authors use the

same objective function as that of the Local Linear Embedding method [57] while restoring the missing entries. Since, in this work, we consider the clustering of data with missing values, next we discuss some methods related to clustering.

For clustering data with missing attributes commonly two types of frameworks are used. One is multi-view and the other is single view clustering. First, we discuss the multi-view clustering. This is a comparatively new approach to handle missing data [16]–[22]. In multi-view, a data point is represented using multiple views. Li *et al*. [17] proposed a multi-view algorithm for incomplete data clustering. Here a multi-view algorithm dealing with subspace mapping and Nonnegative matrix Factorization (NMF) are used to construct a latent subspace. The clustering is done in the latent subspace. Li *et al*. [17], to compare their method also use the method in [16], which is developed for clustering of multi-view complete data, for clustering of data with missing values. For this, they impute the missing values by a base line method before using NMF based clustering. In [18] for each view, imputation is done with the average value of the feature for that view and a new update rule with NMF for each view is used for clustering. In [19] a modified version of [17] is proposed. Here, to find a latent subspace from all views a graph Laplacian term is used. Yin *et al*. [20] also modified [17] to propose a unified method for incomplete unlabeled multi-view data. Here, everything (feature selection, subspace learning, and inter-view and intra-view similarity) necessary for partial multi-view clustering is done jointly to learn a shared representation. Zhao *et al*. [21] use deep semantic mapping with a graph based regulator in the context of multi-view clustering. In [22] authors use an Isomorphic Linear Correlation Analysis (ILCA) method to capture both semantic complementarity and identical distribution among different views of a multi-view data set. Here the authors assume that missing view obeys Gaussian distribution. They propose an efficient algorithm for ILCA and analyze its computational complexity. Next, we discuss some single-view clustering methods for data with missing values.

Hathaway and Bezdek used Fuzzy-c-means (FCM) to cluster data with missing attributes [1]. They proposed four methods. They are whole data strategy (WDS), partial distance strategy (PDS), optimal completion strategy (OCS), and nearest prototype strategy (NPS). In WDS they simply delete all incomplete instances and apply FCM to the remaining complete data. This is useful when the volume of missing data is small. While clustering PDS uses partial distance computed excluding the missing features. The OCS imputes missing data randomly. Here, the missing components are treated as additional variables which are then obtained minimizing the FCM objective functional. NPS is almost the same as the OCS. The only difference here is that during optimization the missing value is replaced by the corresponding feature value of its nearest centroid. In [29] local PCA is used to capture the structure of incomplete data during clustering. Li *et al*. [23] proposed nearest-neighbor intervals to represent missing data. Here a missing value is represented by an interval which is found using the $k-$nearest-neighbors. They assume the FCM centroids as interval-valued vectors and for clustering the distances are computed using intervals. For missing data related to psychological disorder, the performance of four methods in [1] is discussed in [58]. Of the four methods, the OCS method performs the best in that scenario. The authors also show that OCS performs better than regression imputation (RI) and expectation maximization estimation (EME) methods [40]. In [24] genetic algorithms with FCM is applied to cluster data with missing features. Here, each chromosome is defined using some random value within a nearest-neighbor interval which is determined using the available features of the data point with missing inputs. The fitness function of a chromosome is defined as the inverse of the objective function of FCM. To find the missing values, genetic algorithms are applied. After that FCM is applied to it and the whole process is repeated until the optimal result is found. In [25] for each missing attribute authors claim to find q nearest neighbors using partial distances. They also claim to train a BP neural network for each missing attribute. Here the backpropagation learning is used to estimate the missing values and the FCM is applied to cluster the data. From the description provided in [25], it is unclear to us what exactly has been done to deal with the missing attributes. Typically, for clustering data with missing values, before the clustering algorithm is applied, the incomplete data are made complete by imputation. But in [26] instead of using a single value, intervals are used to represent missing data and then they use robust versions of $k-$medians and $k-$means algorithms. Authors in [27] also used intervals to represent missing values but here a robust version of the fuzzy c-means clustering algorithm is used. Datta *et al*. [28] proposed two techniques for clustering data with missing values ($k-$means-Feature Weighted Penalty based Dissimilarity ($k-$means-FWPD) and Hierarchical Agglomerative Clustering-Feature Weighted Penalty based Dissimilarity (HAC-FWPD)). In $k-$means-FWPD, a modified $k-$means algorithm is used for clustering where the distance between two points is measured by the proposed FWPD measure. The HAC-FWPD algorithm is based on hierarchical clustering using the proposed FWPD measure. Here three types of linkage (single, complete and average) algorithms are used for hierarchical clustering.

III. Proposed Algorithms

In this section, the proposed algorithms are presented. Here first we discuss the formulation of our method. Then we discuss the procedure to train an appropriate deep autoencoder network using 50% of a data set. Next we discuss the procedure to represent each incomplete and complete test datum into a latent space generated by a deep autoencoder and clustering of the remaining 50% dataset using the FCM [59] algorithm in the latent space. In the second method, we extend the proposed model so that the latent space representation can preserve the geometric structure (the inter-point distances) of the inputs. To achieve this the objective function of the conventional AE is augmented with the Sammon's stress [39] as a regularizer. Although we use Sammon's stress function here as a regularizer, the underlying concept is quite general and other regularizers can also be used. In fact, use of autoencoders with various constraints on the latent space realized via different regularizers appear to be a promising emerging area of research. The use of SOM-generated quantizers is also an effective way of reducing computational overhead with

a distinct advantage. The self-organizing map enjoys a density matching property and thereby important areas of the input space are much better represented by the SOM quantizers compared to the prototypes generated by conventional clustering algorithms. For example, in [60] Sammon's stress function has been used with an autoencoder for domain adaptation problem.

A. The AutoEncoder Model

We use a multi-layer perceptron (MLP) network to realize an auto-encoder [61]. For simplicity we consider one hidden layer here. Let, p be the number of input features, q be the number of nodes in the hidden layer, $\mathcal{S}(\cdot)$ be the activation function for every node of this network, t_{kj} be the target output corresponding to the j^{th} output neuron for the k^{th} data point, $\mathbf{x}_k$ be the k^{th} data point, n be the number of data points, and r be the number of classes. There is no connection within a layer and there is complete connection between successive layers. In the following we adopt the description used in [62]. For the input layer:

$$\mathcal{S}(x_{ki}) = x_{ki}, i = 1, \ldots, p; \quad \mathcal{S}(x_{k0}) = 1, \quad \forall k; \tag{1}$$

where x_{ki} be the i^{th} component of the input vector $\mathbf{x}_k$ and $\mathcal{S}(x_{k0})$ is the bias neuron signal of the input layer. For the hidden layer:

$$z_{kh} = \sum_{i=0}^{p} w_{ih}\mathcal{S}(x_{ki}), h = 1, \ldots, q$$

$$\mathcal{S}(z_{kh}) = \frac{1}{1+e^{-z_{kh}}}, h = 1, \ldots, q; \quad \mathcal{S}(z_{k0}) = 1, \quad \forall k; \tag{2}$$

where w_{ih} is the weight between i^{th} input node and h^{th} hidden node, w_{0h} is the bias of the h^{th} hidden neuron and $\mathcal{S}(z_0)$ is the signal function of the biased neuron of the hidden layer. For the output layer:

$$y_{kj} = \sum_{h=0}^{q} w_{hj}\mathcal{S}(z_{kh}), j = 1, \ldots, p$$

$$o_{kj} = \mathcal{S}(y_{kj}) = \frac{1}{1+e^{-y_{kj}}}, j = 1, \ldots, p. \tag{3}$$

where w_{hj} is the weight between h^{th} hidden node and j^{th} output node and w_{0j} are the biases of the output neurons. Here, the target is the same as the input. Hence, $t_{kj} = x_{kj}$. So the instantaneous system error for the k^{th} training pattern can be written as follows:

$$E^k = \frac{1}{2}\sum_{j=1}^{p}(x_{kj} - o_{kj})^2. \tag{4}$$

where x_{kj} is the j^{th} feature value of the k^{th} training data point.

B. Training of the Network

Let $X \subseteq R^p$ be the training data. We randomly divide X into two equal halves (equal to the extent possible): $X = X_{TR} \cup X_{TE}, X_{TR} \cap X_{TE} = \emptyset$. We cluster X_{TR} into n_c clusters using a clustering algorithm, say the $k-$means algorithm (note that, here we can use any type of clustering algorithm where the number of clusters is predefined). In this way we produce n_c cluster centers $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n_c}\}; \mathbf{v}_i \subseteq R^p$. Now from X_{TR} we generate p modified data sets $X_{TR}^k, k = 1, 2, \ldots, p$ as follows.

To generate X_{TR}^k for $\mathbf{x}_i \in X_{TR}$, we replace x_{ik} by v_{lk} if $l = \text{argmin}_j\{||\mathbf{x}_i - \mathbf{v}_j||_*^2\}$.

This means that we assume the k^{th} feature value is missing, and the k^{th} feature value is imputed by the k^{th} feature value of the centroid which is closest to $\mathbf{x}_i$ in terms of the distance measure $||.||_*$ computed using all but the k^{th} feature.

We want to generate a latent space representation for X_{TR} as well as $X_{TR}^k, k = 1, 2, \ldots, p$, so that we can use such a representation for clustering of data with missing values.

Now first we train an autoencoder using X_{TR} for N_1 epoches. Next we retrain the same network for N_2 epoches with the data set $X^{Total} = X_{TR} \cup_{k=1}^{p} X_{TR}^k$. For any $\mathbf{x}_i \in X_{TR}$ as well as any $\mathbf{x}_i^k \in X_{TR}^k$ the target vector is taken as $\mathbf{x}_i \in X_{TR}$.

The proposed training method is provided in an algorithmic form as Algorithm 1.

Algorithm 1: Training of the Network.

BEGIN

1: Choose N_1, N_2 and n_c.
2: Randomly divide X into two equal halves X_{TR} and X_{TE} such that $X = X_{TR} \cup X_{TE}, X_{TR} \cap X_{TE} = \emptyset$.
3: Using $k-$Means clustering to cluster X_{TR} into n_c number of cluster and generate n_c cluster centers $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n_c}\}; \mathbf{v}_i \subseteq R^p$.
4: **for** k = 1 to p **do**
5: Generate X_{TR}^k by replacing $\mathbf{x}_{ik}$ by $\mathbf{v}_{lk}$ if $l = \text{argmin}_j\{||\mathbf{x}_i - \mathbf{v}_j||_*^2\}$ where $||.||_*$ computed using all but the k^{th} feature.
6: **end for**
7: We train an autoencoder using X_{TR} for N_1 epoches.
8: Retrain the same network for N_2 epoches with the data set $X^{Total} = X_{TR} \cup_{k=1}^{p} X_{TR}^k$. For any $\mathbf{x}_l^k \in X_{TR}^k$ the target vector is taken as $\mathbf{x}_i \in X_{TR}$.

END

C. Testing Procedure

We use X_{TE} data set for testing. For that first we randomly select data points from X_{TE} with probability of selection $= 0.5$ and in each such selected data point we randomly delete up to $\frac{p}{2}$ features to create $X_{TE'}$. To do this we proceed as follows. For every feature we draw a random no $rand_1$ in [0, 1]. If $rand_1 >= 0.5$ we assume that the corresponding feature is missing. We do this till either $\frac{p}{2}$ features are deleted or all p features are examined. So, for each test data point $\mathbf{x}_i \in X_{TE'}$ up to $\frac{p}{2}$ features may be missing. We impute the missing values of $\mathbf{x}_i$ using $\mathbf{v}_i, (i = 1, \ldots, n_c)$ as follows. The k^{th} missing value of $\mathbf{x}_i$, x_{ik} is imputed by v_{lk} if $l = \text{argmin}_j\{||\mathbf{x}_i - \mathbf{v}_j||_*^2\}$ where $||.||_*$ computed using all but the missing feature value of $\mathbf{x}_i$. We pass $\mathbf{x}_i$ with the imputed values through our trained network and take the output of the hidden layer, $\mathcal{S}(z_{ih})$ as the latent space

Algorithm 2: Testing Procedure.

BEGIN

Randomly select a data point from X_{TE} with probability of selection $= 0.5$ and in each such selected data point we randomly delete up to $\frac{p}{2}$ features to create $X_{TE'}$.
for k $= (\frac{n}{2} + 1)$ to n **do**
  Let, $\mathbf{x}_k \in X_{TE'}$.
  for j $= 1$ to p **do**
    if x_{kj} is missing **then**
      Impute $x_{kj} = v_{lj}$ if $l = \text{argmin}_i\{||\mathbf{x}_k - \mathbf{v}_i||_*^2\}$ where $||.||_*$ computed using all but the missing feature of $\mathbf{x}_k$.
    end if
  end for
end for
$\widetilde{Z} = \emptyset$.
for i$=(\frac{n}{2}+1)$ to n **do**
  Let, $\mathbf{x}_i \in X_{TE'}$.
  Calculate $\mathcal{S}(\mathbf{z_i})$ using eq. (5).
  $\widetilde{Z} = \widetilde{Z} \cup \mathcal{S}(\mathbf{z_i})$.
end for
Cluster X_{TE} by FCM for r number of clusters and store the partition matrix in $\mathbf{CLUS_1}$.
Cluster $\widetilde{Z}$ by FCM with the same initial partition matrix for r number of clusters and store the partition in $\mathbf{CLUS_2}$.
Find out the similarity between $\mathbf{CLUS_1}$ and $\mathbf{CLUS_2}$ with NMI and ARI.

END

representation:

$$z_{ih} = \sum_{k=0}^{p} w_{kh}\mathcal{S}\left(x_{ik}\right), h = 1, \ldots, q$$
$$\widetilde{z}_{ih} = \mathcal{S}\left(z_{ih}\right) = \frac{1}{1 + e^{-z_{ih}}}, h = 1, \ldots, q. \quad (5)$$

Thus, we get a latent space representation of X_{TE} as $\widetilde{Z} = \{\widetilde{\mathbf{z}}_1, \ldots, \widetilde{\mathbf{z}}_{\frac{n}{2}}\}$; $\widetilde{Z} \subseteq R^q$. Now, we apply the FCM [59] on X_{TE} to find r number of clusters. With the same initial partition matrix, we apply FCM on $\widetilde{Z}$ for the same number of clusters. Thus, we cluster X_{TE} which does not have any missing values and we also cluster $\widetilde{Z}$ which has been generated after injecting missing values in $X_{TE'}$. If our method of handling missing values for the purpose of clustering is good then both X_{TE} and $\widetilde{Z}$ should produce similar cluster structure. To assess this we try to find the similarity between the clusters created from the two datasets using two popular proximity measures, normalized mutual information (NMI) [36] and adjusted rand index (ARI) [37]. We repeat the entire process five times and report the average values of NMI and ARI.

The proposed testing method is provided in an algorithmic form as Algorithm 2.

D. Structure-Preserving Autoencoder for Handling Missing Values

Typically, an AE produces an encoding that can reproduce the data. It does not care about the "geometric structure" present in the data. Although, it is hard to formally define what we mean by "structure," we want to preserve, for example, the cluster substructures present in the input in the latent space representation. If we can achieve this, we expect the latent space representation to perform well with any distance-based algorithm. Since our final goal is to cluster a data set with missing information, if the inter-point distances of the inputs are preserved in the latent space representation, we expect the projected data to maintain the same/similar cluster structure as that of the input data. In fact, even a distance-based classification algorithm is expected to perform well with the latent space representation. In order to realize this we use the Sammon's stress function [39] as a regularizer to the original objective function of the autoencoder.

Let X_{TR} be the training data, $n = |X_{TR}|; \mathbf{x}_i \in X_{TR}; i = 1, \ldots, n$. Let $Z_{TR}; \mathbf{z}_i \in Z_{TR}; i = 1, \ldots, n$ be the latent space representation of X_{TR}. The Sammon's stress between X_{TR} and Z_{TR} is defined as

$$E_S(X_{TR}; Z_{TR}) = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=1, j>i}^{n} \frac{(d_{ij}^* - d_{ij})^2}{d_{ij}^*};$$
$$\text{where} \quad d_{ij}^* = ||\mathbf{x}_i - \mathbf{x}_j||; d_{ij} = ||\mathbf{z}_i - \mathbf{z}_j||;$$
$$i = 1, 2, \ldots, n; j = 1, 2, \ldots, n; \quad (6)$$

where $\mathbf{x}_i \in X_{TR}$ and $\mathbf{z}_i \in Z_{TR}$; $\mathbf{z}_i$ is the latent space representation of $\mathbf{x}_i; i = 1, 2, \ldots, n$.

Now the objective function of regularized AE is defined as:

$$E = \frac{1}{2} \sum_{k=1}^{n} ||\mathbf{x}_k - \mathbf{o}_k||^2 + \lambda E_S. \quad (7)$$

where $\mathbf{o}_k$ is the autoencoder output corresponding to $\mathbf{x}_k$, E_S is the Sammon's stress and λ is the multiplicative factor of the regularizer.

If we use the above objective function, while updating the network, after every update (for online learning, once for every data point; for mini batch, once after every mini batch) we have to compute $n(n-1)/2$ distances, which makes the algorithm computationally very expensive. To reduce the computational overhead, we quantize X_{TR} into N prototypes, $V' = \{\mathbf{v'_1}, \mathbf{v'_2}, \ldots, \mathbf{v'_N}\}$, using a self-organizing feature map (SOM) [63]. Instead of preserving the inter-point distances over the entire training data set, we do so on the set of prototypes V' extracted by the SOM. We prefer SOM over clustering algorithms because SOM has a density matching property and as a result, corresponding to any densely populated area of the input space, there will be more prototypes. Hence, the important areas of the input space (as represented by the training data) will be better preserved. So we augment the training data X_{TR} with V' and compute the regularizer only using V'. Thus, the new

regularizer becomes:

$$E_V(V';Z_V) = \frac{2}{N(N-1)} \sum_{i=1}^{N-1} \sum_{j=1,j>i}^{N} \frac{(d'^{*}_{ij} - d'_{ij})^2}{d'^{*}_{ij}};$$
$$\text{where} \quad d'^{*}_{ij} = ||\mathbf{v}'_i - \mathbf{v}'_j||; d'_{ij} = ||\mathbf{z}'_i - \mathbf{z}'_j||;$$
$$i = 1, 2, \ldots, N; j = 1, 2, \ldots, N; \quad (8)$$

where Z_V is the latent space representation of $V'; \mathbf{z}'_i \in Z_V; \mathbf{z}'_i$ is the hidden layer output of $\mathbf{v}'_i; i = 1, 2, \ldots, N$. The new AE objective function becomes:

$$E = \frac{1}{2}\left(\sum_{k=1}^{n} ||\mathbf{x}_k - \mathbf{o}_k||^2 + \sum_{k=1}^{N} ||\mathbf{v}'_k - \mathbf{o}'_k||^2\right) + \lambda E_V. \quad (9)$$

$\mathbf{o}'_k$ is the autoencoder output corresponding to $\mathbf{v}'_k$.

In the first phase of the training, we train the autoencoder using $\widetilde{X} = X_{TR} \cup V'$ and in the second phase of training we train the autoencoder using $X^{Total} = \widetilde{X} \cup_{k=1}^{p} X_{TR}^k$. In the first phase the targets are the same as the corresponding inputs and in the second phase of training for any $\mathbf{x}_i^k \in X_{TR}^k$ the target vector is taken as the corresponding $\mathbf{x}_i \in X_{TR}$.

IV. Experiments

A. Data Sets

We take 12 data sets from the well-known University of California at Irvine (UCI) repository [64]. For each data set, the instances are randomly shuffled. From each shuffled dataset, we take first 50% points for training and remaining 50% points for testing. From the testing points, we randomly remove up to $(\frac{p}{2})$ features from each of the data points. This procedure to create the missing dataset is the same procedure as in [28] but we use only 50% data as the remaining 50% is used as the training data. We note that the methods in [28] did not need such training data. The Algorithm-S-I (an algorithm/a table number with a prefix 'S-' indicates that the algorithm/table has been placed in the Supplementary Materials) describes the missing dataset creation method in an algorithmic form. The summary of the 12 datasets used is listed in Table S-I.

B. Experimental Set Up

Based on few trial and error experiments, for all data sets we have used the following architecture of the deep autoencoder network: $(p) - (4 \times p) - (3 \times p) - (2 \times p) - (3 \times p) - (4 \times p) - (p)$ where p is the number of features in the corresponding dataset. For example, if a data set has 4 features, then in the input layer as well as in the output layer the number of nodes are 4, in the first and fifth hidden layers the number of nodes are $(4 \times 4) = 16$, in second and forth hidden layers the number of nodes are $(3 \times 4) = 12$ and in third hidden layer the number of nodes is $(2 \times 4) = 8$. So the network architecture is $(4 - 16 - 12 - 8 - 12 - 16 - 4)$. We do not claim that this is the best/optimal architecture, particularly when p is large and n is small.

The next important issue is to decide on the hidden layer whose output should be used for the latent space representation. Theoretically speaking output of any of the layers, 2^{nd} through 6^{th}, can be used. We handle this problem in two ways. For the cardio dataset, we use a bi-level cross-validation to decide on the hidden layers for the latent space representation. For this, we proceed as follows. Let $X = X_{TR} \cup X_{TE}$ as divided earlier. Now we randomly divide X_{TR} into two equal parts as $X_{TR} = X_{TR_1} \cup X_{TR_2}, X_{TR_1} \cap X_{TR_2} = \emptyset$. We train an autoencoder using Algorithm 1 on X_{TR_1} with $N_1 = N_2 = 500$ iterations and test using Algorithm 2 with X_{TR_2} as test data. Here we first perform clustering on X_{TR_2}. Let the partition on X_{TR_2} be U_1. Then using the same initial condition we perform clustering three times separately using the output of the first, second and third hidden layers as the latent space representation. Let the three partitions be UH_1, UH_2 and UH_3. Now to compute the similarity between U_1 and $UH_i \; \forall i = 1, 2, 3$ we compute $NMI_i^1 = NMI(U_1, UH_i; i = 1, 2, 3)$ and $ARI_i^1 = ARI(U_1, UH_i; i = 1, 2, 3)$. Next, we switch the roles of X_{TR_1} and X_{TR_2} and repeat the experiment. In this way we get NMI_i^2 and $ARI_i^2; i = 1, 2, 3$. Now we find $k = \max_i(\frac{NMI_i^1 + NMI_i^2}{2})$. Thus, in terms of NMI for X_{TR} the best choice for latent space representation is the k^{th} layer, $k \in \{1, 2, 3\}$. Now we train an autoencoder using Algorithm 1 on X_{TR} and test the performance of the k^{th} hidden layer representation using Algorithm 2 with X_{TE}. We do the same process to find ARI for X_{TR}. We then switch the roles of X_{TR} and X_{TE}. The entire process is repeated 5 times and the average values of ARI and NMI are reported. Table S-II and Table S-III show the detailed results of the cross-validation experiments for the Cardio data set. From Table S-II we find that with NMI from the 10 experiments 8 experiments selected the first hidden layer; while with ARI in every case the first hidden layer is selected.

Thus, for all our remaining experiments with other data sets, we use the output of the first hidden layer as the latent space representation. To train a network for each data set we take the learning rate $\eta = 0.9$, $n_c = (r \times 3)$. If no label information is available, we can choose n_c either using the domain knowledge or by using some cluster validity indices [65], [66]. For the FCM algorithm we take the fuzzy exponent $m = 2$. The, m in FCM is called the fuzzification parameter. It controls the degree of fuzziness in the partition matrix. FCM $\rightarrow k-$means as $m \rightarrow 1^+$. Also as $m \rightarrow \infty$, the membership of every point to every cluster becomes equal to $(\frac{1}{r})$ where r is the number of clusters. In other words, we get the most fuzzy partition. To reduce the computational overhead we vary the value of N_1 and N_2 for each data set. As the sizes of Cardio, Chronic Kidney, Libras, and Pendigits are bigger than those of others we choose $N_1 = N_2 = 1000$ for these datasets. For the remaining 8 datasets we choose $N_1 = N_2 = 10000$. For each of the 12 datasets we generate five random training-test partitions. For each partition, we train the autoencoder 10 times. For each latent space representation, we run the FCM algorithm for 10 different initial partitions and report the average NMI [36] and ARI [37].

C. Performance Evaluation Measures

To evaluate the performance of our missing data handling algorithm for the clustering problem, we should compare the

clusters found using the data with missing values and the clusters found using the complete data. For this we use two popular indices NMI [36] and ARI [37]. Let, $U_{r\times n}$ and $V_{r\times n}$ be two partitions obtained by clustering a data set with missing information and the original data respectively. We define U as $U(i,j)=1$ if $\mathbf{x}_j \in i^{th}\ cluster$ and $U(i,j)=0\ otherwise$. Similarly, we define V as $V(i,j)=1$ if $\mathbf{x}_j \in i^{th}\ cluster$ and $V(i,j)=0\ otherwise$. Let the clusters corresponding to U be $U_1,\ldots,U_r$ and clusters corresponding to V be $V_1,\ldots,V_r$. Note that U and V may have different number of clusters, but here they are the same. Let N_i be the number of points in cluster U_i and M_j be the number of points in cluster V_j. We denote N_{ij} as the number of points that are common between U_i and V_j. The ARI [37] is defined in Eq. (10) shown at the bottom of this page, and the NMI between two random variables X and Y is defined in (11).

$$NMI(X,Y)=\frac{I(X,Y)}{\sqrt{H(X)H(Y)}}. \tag{11}$$

In (11), $I(X,Y)$ denotes the mutual information between X and Y, $H(X)$ denotes the entropy of X and $H(Y)$ denotes the entropy of Y. We have to find the NMI between U and V. To compute $NMI(U,V)$, first we calculate the entropy of partition U, $H(U)$ and entropy of partition V, $H(V)$. To compute the mutual information $I(U,V)$ between partitions U and V, first we calculate the joint entropy $H(U,V)$ between U and V and calculate $I(U,V)$ as: $I(U,V)=H(U)+H(V)-H(U,V)$. The detailed procedure of computing $H(U)$, $H(V)$, and $H(U,V)$ is as below:

Define $U_{P_i}=\frac{1}{n}\sum_{k=1}^{n}U(i,k)$. So, $H(U)=-\sum_{i=1}^{r}U_{P_i}\ log_2(U_{P_i})$. Similarly, we compute the entropy of the partition V, $H(V)$.

Now, to find the joint entropy $H(U,V)$ between partitions U and V, we create another matrix Q as $Q=(UV^T)$ and compute $H(U,V)=-\frac{1}{n}\sum_{i=1}^{r}\sum_{k=1}^{r}Q(k,i)log_2(Q(k,i))$. Now by using $I(U,V)$, $H(U)$ and $H(V)$ we compute $NMI(U,V)$ with (11).

D. Experimental Results

Table I compares the results of our method along with five other methods in terms of NMI and ARI. Here, we use the results as presented in [28] to compare our methods. We note here that the experimental protocol in [28] is not exactly the same as that of our method. From Table I we see that for 9 of the 12 data sets the performance of our method is the best among the six algorithms with respect to NMI and ARI. HAC-FWPD performs the best for Pendigit and Libras data sets. For the Iris data set the performance of SVDI is the best.

To check the effect of various parameters of our method we vary them and check their influence on the performance. To check the effect of N_2, we increased N_2 to 40000 and experimented with three data sets. Table II summarizes the results. From Table II we find that increasing N_2 to 40000 from 10000 practically has no effect on the performance. For Table I, as mentioned earlier, we have used the output of the first hidden layer as the latent space representation. This raises three related questions: why the first layer? Can we use the outputs of other layers to improve the performance? How does the performance change if we use two-level cross-validation to decide on the hidden layer to be used for the latent space representation? As discussed earlier, to address these issues we use two level two-fold cross-validations to decide on the hidden layer. Because of the computational overhead we do this experiment only for the Cardio data and repeat it five times. Table S-II provides the detailed outputs for each fold and for each repetition of the inner level. There are a total of twenty cases (5 repeats $\times 2$ folds $\times 2$ indices). Of these 20 cases in 18 cases, cross-validation suggested the use of the first hidden layer and in the remaining two cases the second hidden layer. Hence, in all our experiment we have used the output of the first hidden layer for the latent space representation. A very plausible reason for the choice of the first hidden layer is that the output of the first hidden layer will preserve more information of the input than that by each of the other hidden layers. Table S-III shows the output of the outer level. It also reports the average values of NMI and ARI.

Here we use 50–50 partition with repetition of five times for doing each experiment. The 50–50 partition is just one of the ways of doing the experiment. There is no specific reason behind the use of a 50–50 partition. The method is quite general. For example, one can use a 10 fold partition.

In Table S-IV we report the results on 12 UCI datasets by choosing the first hidden layer of a five-hidden layer autoencoders and of a one-hidden-layer autoencoder. From the results we find that when we use a five-hidden layer network the performance improves both in terms of NMI [36] and ARI [37] in most of the cases. In particular, using NMI in 11 of the 12 cases the 7-layer architecture performs better than the 3-layer architecture while in terms of ARI in 10 cases the 7-layer architecture performs better. In several cases, the improvement with the 7 layer network is significantly high. A possible reason for this may be the following. When we use 5 hidden layers, the output of every layer can be a latent representation. So the first hidden layer needs to make a good representation that preserves sufficient information to make all higher level representation good enough to reproduce the input.

In Table III we compare results of the 12 UCI data sets using FCM with $m=2$ and the $k-$means algorithm on the latent

$$ARI(U,V)=\frac{\sum_{i,j}\binom{N_{ij}}{2}-\left[\sum_i\binom{N_i}{2}\sum_j\binom{M_j}{2}\right]\Big/\binom{n}{2}}{\frac{1}{2}\left[\sum_i\binom{N_i}{2}+\sum_j\binom{M_j}{2}\right]-\left[\sum_i\binom{N_i}{2}\sum_j\binom{M_j}{2}\right]\Big/\binom{n}{2}} \tag{10}$$

TABLE I
RESULTS OF THE 12 UCI DATASETS AND COMPARISON TO OTHER METHODS. THE VALUES IN PARENTHESES (.) ARE THE STANDARD DEVIATIONS

Dataset	Measure	Our	HAC - FWPD	ZI	MI	SV DI	kN NI
Balance	NMI	**0.21 (0.02)**	0.12 (0.06)	0.05 (0.05)	0.06 (0.05)	0.10 (0.02)	0.08 (0.04)
	ARI	**0.21 (0.02)**	0.12 (0.07)	0.05 (0.05)	0.06 (0.05)	0.10 (0.02)	0.08 (0.04)
Cardio	NMI	**0.80 (0.03)**	0.43 (0.17)	0.34 (0.10)	0.34 (0.11)	0.35 (0.09)	0.02 (0.01)
	ARI	**0.77 (0.03)**	0.50 (0.19)	0.37 (0.11)	0.36 (0.12)	0.38 (0.09)	0.02 (0.01)
Chronic Kidney	NMI	**0.76 (0.04)**	0.71 (0.36)	0.65 (0.47)	0.65 (0.47)	0.65 (0.47)	0.00 (0.00)
	ARI	**0.81 (0.05)**	0.71 (0.36)	0.67 (0.47)	0.67 (0.47)	0.67 (0.47)	0.00 (0.00)
Glass	NMI	**0.71 (0.10)**	0.66 (0.09)	0.54 (0.12)	0.58 (0.12)	0.56 (0.13)	0.13 (0.10)
	ARI	**0.81 (0.08)**	0.74 (0.09)	0.57 (0.16)	0.62 (0.15)	0.61 (0.17)	0.05 (0.07)
Iris	NMI	0.79 (0.07)	0.88 (0.07)	0.89 (0.06)	0.89 (0.05)	**0.90 (0.06)**	0.70 (0.02)
	ARI	0.78 (0.08)	0.92 (0.05)	0.93 (0.05)	0.94 (0.05)	**0.94 (0.03)**	0.56 (0.03)
Libras	NMI	0.57 (0.06)	**0.88 (0.03)**	0.61 (0.03)	0.65 (0.06)	0.69 (0.05)	0.44 (0.05)
	ARI	0.54 (0.08)	**0.84 (0.05)**	0.25 (0.03)	0.31 (0.10)	0.41 (0.13)	0.14 (0.05)
Monk1	NMI	**0.33 (0.04)**	0.16 (0.14)	0.03 (0.03)	0.00 (0.00)	0.01 (0.02)	0.01 (0.01)
	ARI	**0.39 (0.05)**	0.19 (0.15)	0.01 (0.01)	0.00 (0.00)	0.02 (0.02)	0.00 (0.01)
Monk2	NMI	**0.44 (0.07)**	0.21 (0.19)	0.01 (0.02)	0.03 (0.05)	0.20 (0.20)	0.01 (0.01)
	ARI	**0.51 (0.07)**	0.25 (0.21)	0.00 (0.00)	0.02 (0.04)	0.25 (0.25)	0.01 (0.02)
Monk3	NMI	**0.50 (0.07)**	0.18 (0.19)	0.02 (0.02)	0.02 (0.03)	0.07 (0.12)	0.03 (0.03)
	ARI	**0.58 (0.07)**	0.22 (0.21)	0.00 (0.00)	0.01 (0.01)	0.09 (0.15)	0.01 (0.02)
Pendigit	NMI	0.67 (0.03)	**0.78 (0.09)**	0.23 (0.19)	0.23 (0.21)	0.25 (0.19)	0.39 (0.15)
	ARI	0.49 (0.04)	**0.80 (0.09)**	0.43 (0.20)	0.43 (0.21)	0.57 (0.26)	0.60 (0.16)
Seeds	NMI	**0.78 (0.07)**	0.54 (0.11)	0.51 (0.03)	0.49 (0.06)	0.52 (0.09)	0.64 (0.06)
	ARI	**0.80 (0.06)**	0.49 (0.16)	0.40 (0.04)	0.37 (0.07)	0.46 (0.13)	0.64 (0.11)
Vehicle	NMI	**0.82 (0.02)**	0.72 (0.12)	0.54 (0.21)	0.47 (0.18)	0.60 (0.13)	0.16 (0.01)
	ARI	**0.78 (0.04)**	0.75 (0.14)	0.46 (0.30)	0.32 (0.26)	0.57 (0.25)	0.09 (0.01)
Cardio (Bi-label cross-validation)	NMI	**0.78 (0.00)**	0.43 (0.17)	0.34 (0.10)	0.34 (0.11)	0.35 (0.09)	0.02 (0.01)
	ARI	**0.89 (0.01)**	0.50 (0.19)	0.37 (0.11)	0.36 (0.12)	0.38 (0.09)	0.02 (0.01)

TABLE II
PERFORMANCE OF THE PROPOSED METHOD WITH DIFFERENT N_1 AND N_2 ON 2 UCI DATA SETS. THE VALUES IN PARENTHESES (.) ARE THE STANDARD DEVIATIONS

Dataset	N_1	N_2	NMI	ARI
Balance	10000	40000	0.21(0.01)	0.22(0.01)
Iris	10000	40000	0.79(0.05)	0.78(0.07)
Seeds	10000	40000	0.78(0.06)	0.80(0.05)

TABLE III
COMPARISON OF RESULTS USING DIFFERENT CLUSTERING ALGORITHMS IN THE LATENT SPACE REPRESENTATION. THE VALUES IN PARENTHESES (.) ARE THE STANDARD DEVIATIONS

Dataset	FCM ($m = 2$)		$k-$means	
	NMI	ARI	NMI	ARI
Balance	0.21(0.02)	0.21(0.02)	0.21(0.05)	0.21(0.05)
Cardio	0.80(0.03)	0.77(0.03)	0.88(0.03)	0.91(0.04)
Chronic Kidney	0.76(0.04)	0.81(0.05)	0.78(0.04)	0.81(0.05)
Glass	0.71(0.10)	0.81(0.08)	0.74(0.12)	0.72(0.11)
Iris	0.79(0.07)	0.78(0.08)	0.78(0.06)	0.77(0.08)
Monk1	0.33(0.04)	0.39(0.05)	0.30(0.03)	0.35(0.03)
Monk2	0.44(0.07)	0.51(0.07)	0.35(0.02)	0.39(0.02)
Monk3	0.50(0.07)	0.58(0.07)	0.56(0.07)	0.64(0.06)
Pendigit	0.67(0.03)	0.49(0.04)	0.82(0.00)	0.80(0.00)
Seeds	0.78(0.07)	0.80(0.06)	0.77(0.07)	0.78(0.07)
Vehicle	0.82(0.02)	0.78(0.04)	0.80(0.06)	0.74(0.08)
Libras	0.57(0.06)	0.54(0.08)	0.76(0.15)	0.72(0.12)

TABLE IV
DEMONSTRATION OF THE EFFECT OF THE SECOND PHASE OF THE TRAINING USING THE AUGMENTED DATA ON 2 UCI DATA SETS. THE VALUES IN PARENTHESES (.) ARE THE STANDARD DEVIATIONS

Dataset	Proposed	Proposed	Complete	Complete
	NMI	ARI	NMI	ARI
Glass	0.71 (0.10)	0.81 (0.08)	0.70 (0.06)	0.64 (0.09)
Seeds	0.78 (0.07)	0.80 (0.06)	0.78 (0.06)	0.80 (0.05)

space representation of the incomplete dataset. From this table, we found that in terms of both NMI and ARI out of 12 cases FCM wins in 6 cases and $k-$means wins in 6 cases. So, the two algorithms performed similarly in terms of NMI and ARI at least for this problem. But as FCM is a generalized version of $k-$means (FCM approaches the $k-$means as $m \to 1^+$), we use FCM as for some data with overlapped cluster structure, FCM could be more effective.

In order to see the effectiveness of our two-stage training, we conduct experiments on two data sets: Seed and Glass. In the second phase of training instead of using the augmented data set (where we deliberately inserted missing values and imputed them using a $k-$NN based method), we use the original data to further train the autoencoder. In Table IV we compare the results. Table IV reveals that in the case of Glass, the proposed method performs much better while for the other data set, the performance remains the same.

Most of the systems trained with autoencoders use latent spaces of dimension less than that of the input, i.e., use a bottleneck layer. But in this study, we use a higher number of

TABLE V
COMPARISON OF RESULTS ON TWO UCI DATA SETS BY CHOOSING DIFFERENT NUMBER OF NODES IN THE HIDDEN LAYERS WHICH ARE LESS THAN THE INPUT DIMENSION. THE VALUES IN PARENTHESES (.) ARE THE STANDARD DEVIATIONS

Dataset	p-4p-3p-2p-3p-4p-p		p-4-3-2-3-4-p	
	NMI	ARI	NMI	ARI
Glass	0.71(0.10)	0.81(0.08)	0.49(0.03)	0.38(0.05)
Seeds	0.78(0.07)	0.80(0.06)	0.60(0.02)	0.60(0.02)

TABLE VI
PERFORMANCE OF THE PROPOSED METHODS WITH SAMMON'S STRESS AS REGULARIZER. THE VALUES IN PARENTHESES (.) ARE THE STANDARD DEVIATIONS

Dataset	NMI		ARI	
	(Without regularizer)	(With regularizer)	(Without regularizer)	(With regularizer)
Balance	0.21 (0.02)	0.42 (0.10)	0.21 (0.02)	0.45 (0.12)
Glass	0.71 (0.10)	0.83 (0.04)	0.81 (0.08)	0.82 (0.06)
Iris	0.79 (0.07)	0.82 (0.08)	0.78 (0.08)	0.81 (0.10)
Monk1	0.33 (0.04)	0.75 (0.09)	0.39 (0.05)	0.83 (0.09)
Monk2	0.44 (0.07)	0.95 (0.03)	0.51 (0.07)	0.97 (0.02)
Monk3	0.50 (0.07)	0.85 (0.07)	0.58 (0.07)	0.91 (0.05)
seeds	0.78 (0.07)	0.89 (0.07)	0.80 (0.06)	0.90 (0.06)
Vehicle	0.82 (0.02)	0.90 (0.07)	0.78 (0.04)	0.88 (0.08)
Cardio	0.80 (0.03)	0.92 (0.02)	0.77 (0.03)	0.94 (0.01)
Chronic Kidney	0.76 (0.04)	0.82 (0.05)	0.81 (0.05)	0.86 (0.06)
Libras	0.57 (0.06)	0.86 (0.02)	0.54 (0.08)	0.74 (0.06)
Pendigit	0.67 (0.03)	0.90 (0.07)	0.49 (0.04)	0.88 (0.08)

hidden nodes than that in the input layer. Unlike the problem of classification, where the main objective is to achieve good generalization on the test data, here our objective is to partition a given data set and hence a faithful reconstruction of the inputs is of great importance. This is one of the reasons for using more nodes in the hidden layer than the number of inputs. Here, we illustrate its effectiveness using an experiment. In Table V we compare results on 2 UCI data sets (Glass ($p = 9$) and Seeds ($p = 7$)) using our proposed method and the same system trained with $p - 4 - 3 - 2 - 3 - 4 - p$ architecture where p is the number of features in the corresponding dataset. Note that, there is no special reason to choose $p - 4 - 3 - 2 - 3 - 4 - p$ architecture. Here we have taken the number of nodes in the hidden layer less than the number of nodes in the input layer. For both Glass and Seeds datasets the performance of the proposed method $(p - 4p - 3p - 2p - 3p - 4p - p)$ is found to be better than that of the system trained with $p - 4 - 3 - 2 - 3 - 4 - p$ architecture. In both datasets and in both experiments $N_1 = N_2 = 10000$, the latent space representation is taken from the first hidden layer, and each experiment is repeated five times.

E. Results by the Structure-Preserving Autoencoder-Based Method

The results of the method are reported in Table VI . We repeated the experiments five times with ten different weight initializations and report the average results in Table VI. For each data set, we have used a (7×7) SOM to find the prototypes. We have used only those SOM prototypes which have become winner for at least one training instance. In Table VI we have compared the results of this structure-preserving method with those of the first method. From Table VI we find that for all datasets the performance of the proposed method with regularizer is better than that of method without regularizer both in terms of NMI and ARI. For some of the data sets, the improvement is quite significant. A possible reason behind this is that the regularized method tries to maintain the inter-point distances between the prototypes and their latent space representations. Consequently, the cluster structure present in the input data is better preserved in the latent space representation. Based on our limited experiments, we find that if the dimensionality of a data set is high, then E_V is likely to have a higher value and in that case, use of a lower penalty factor λ is found to be more effective to have a smooth convergence of the algorithm. We note here that in the original Sammon's stress, the objective function is divided by $\sum_i \sum_{j>i} d^*_{ij}$. Since, this quantity is a constant for a given data set, theoretically, it does not have any effect when it is optimized independently. When it is used as a regularizer, it scales the penalty coefficient. Hence, we have not considered it as a part of the penalty. However, use of this term in the penalty will reduce the impact of the dimension of the data set and may enable us to use the same value of λ for all data sets. Due to lack of time and computational resources, we could not do this experiment. In the present investigation, for the first eight data sets (which are of lower dimension) we use $N_1 = N_2 = 500; \eta = 0.9; \lambda = 0.01$ and for the remaining data sets, we use $N_1 = N_2 = 50; \eta = 0.9; \lambda = 0.0001$.

F. Parameter Sensitivity

In this section, we try to illustrate the effect of different number of nodes in each hidden layer and the value of m in the FCM algorithm on the performance of the proposed method. Since all our data sets are classification data, we use the number of clusters = the number of classes. In a general setting, like any exploratory data analysis using clustering, if we have domain knowledge we may be able to choose the number of clusters. Otherwise, we need to use some cluster validity index to decide on the number of clusters. Like any iterative clustering method that depends on initialization if the number of clusters does not match with the number of natural substructures present in the data, the performance may vary noticeably with the initialization.

Next we try to illustrate the effect of the number of nodes in each hidden layer on the performance of the proposed method. Due to computational overhead here also we vary the number of nodes for two data sets (Glass and Seeds). We consider three architectures with different nodes in different hidden layers as A_1, A_2 and A_3 : $A_1 = p - 4p - 3p - 2p - 3p - 4p - p$; $A_2 = p - (4p - 5) - (3p - 4) - (2p - 3) - (3p - 4) - (4p - 5) - p$, $A_3 = p - (4p + 5) - (3p + 4) - (2p + 3) - (3p + 4) - (4p + 5) - p$. Note that, there is no special reason to choose the given architectures. Here we have used five different two-fold (Training- Test) partitions. For each training-test partition and each architecture we train 10 different deep autoencoders. We use the output of the first hidden layer and run the FCM

TABLE VII
PERFORMANCE OF THE PROPOSED METHOD WITH DIFFERENT ARCHITECTURES ON 2 UCI DATA SETS. THE VALUES IN PARENTHESES (.) ARE THE STANDARD DEVIATIONS

Dataset		A_1	A_2	A_3
Glass	NMI	0.71(0.10)	0.64(0.04)	0.69(0.05)
	ARI	0.81(0.08)	0.56(0.04)	0.62(0.07)
Seeds	NMI	0.78(0.06)	0.77(0.05)	0.78(0.06)
	ARI	0.80(0.05)	0.79(0.04)	0.81(0.05)

clustering algorithm 10 times (10 different initializations of the FCM partition) using $N_1 = N_2 = 10000$ for training each autoencoder. For each case, we compute the ARI and NMI. Table VII depicts the average NMI and ARI for two datasets for all three architectures, A_1, A_2 and A_3. Table VII reveals that for the Glass dataset if we increase or decrease the number of hidden nodes the performance is decreased. But in the case of Seeds dataset if we increase the number of hidden nodes the performance increases and if we decrease the number of hidden nodes the performance decreases. Thus, we can say that for every data set there may be an optimal architecture. If we can select the optimal architecture, say by cross-validation, the proposed scheme is likely to perform even better. If one has sufficient computing power, in principle, the desirable architecture may be chosen using two-level cross-validation experiments.

Now we try to illustrate the effect of the value of m in FCM on the performance of the proposed method. Here we consider the following six values of m: $2, 1.02, 1.05, 1.08, 1.2, 1.4$. There is no special reason to choose this set of values. Here also we have used five different two-fold (Training- Test) partitions. For each training-test partition and each architecture, we train 10 different deep autoencoders. For Cardio, Chronic Kidney, Libras, Pendigits datasets we use $N_1 = N_2 = 1000$ iterations, while for the remaining data sets we use $N_1 = N_2 = 10000$ iterations. We use the output of the first hidden layer and run the FCM clustering algorithm 10 times (10 different initializations of the FCM partition). Table S-V and Table S-VI, respectively show the NMI and ARI obtained by our method for all datasets with different values of m. In terms of NMI (Table S-V), we find that for Balance and Seeds data sets the performance more or less remains the same for all m. But for the remaining 10 datasets with changes in m, the performance both increases and decreases. There is no consistent behavior. Now if we consider ARI, it is seen that only for Glass dataset if we change the value of m the performance decreases form that with $m = 2$. For the Balance dataset, the performance changes a little bit, but for the remaining data sets the behavior of performances is like that of NMI, i.e, the performance both increases and decreases.

If we consider the NMI and ARI for the best performing m for each data set in Table I, then except three cases (two cases for Iris and one case for Libras) our method performs the best. This investigation suggests that for each data set there may exist an optimal choice of m, which may be exploited using cross-validation experiment, provided we have sufficient computing power.

G. Experiment With Denoising Autoencoder

Can the performance of the proposed method be improved using denoising autoencoder? To check the performance of the proposed method using denoising autoencoder we did some experiments using denoising autoencoders.

In [67] the authors describe the denoising autoencoder as a special type of autoencoder where the input is first corrupted and the autoencoder is trained using the corrupted data as input and uncorrupted input as the target. However, one may say that the proposed method is a special type of denoising autoencoder. But there are some significant differences. In the proposed method first, we train an autoencoder with complete data for some iterations and then we delete each feature once and judiciously impute it. Then we refine the network again for some iterations using both the uncorrupted data and the imputed data. Here also the target vector is the same as the original input as used in denoising autoencoder.

This procedure explained here is almost the same as that of the proposed method. But the difference is that in the training phase after creating X_{TR} and $X^{Total} = X_{TR} \cup_{k=1}^{p} X_{TR}^k$ we create $X'_{TR} = X_{TR} \cup X''_{TR}$ where X''_{TR} is a corrupted version of X_{TR} formed with 10% and 15% gaussian noise with 0 mean and σ standard deviation. Here, first we train an autoencoder using X'_{TR} for N_1 epochs. Next we retrain the same network for N_2 epochs with the data set $X^{Total} = X_{TR} \cup_{k=1}^{p} X_{TR}^k$. For any $\mathbf{x}_i \in X'_{TR}$ as well as any $\mathbf{x}_i^k \in X_{TR}^k$ the target vector is taken as $\mathbf{x}_i \in X_{TR}$. Because of our limited computational resources, we do this experiment only for the Balance, Glass, and seeds data sets and repeat it five times. Table S-VII provides the detailed results for NMI and ARI of these experiments.

For each data set, we have considered four different choices of σ. Note that there is no special reason for choosing these σ. We simply wanted to vary σ from low to high to check the effect of σ on the performance. For the Balance data set, the performance of the denoising autoencoder version of the algorithm is better than the original algorithm both in terms of NMI and ARI for 10% Gaussian noise; but for 15% noise (Table S-VII) it is worse than the original algorithm both in terms of NMI and ARI. For Seeds, on the other hand, the proposed method performed better than all four cases with denoising autoencoders (both for 10% and 15% noise). In the case of Glass, which does not have a clear cluster structure in the data, in terms of NMI the denoising autoencoder version performed better while in terms of ARI, our original algorithm performed better (both for 10% and 15% noise). Based on this limited experiment we cannot infer whether our algorithm with the denoising autoencoder can help or not.

H. Statistical Significance: Wilcoxon Signed Ranks Test

We have performed Wilcoxon signed rank tests (1-tailed) to compare the proposed method with HAC-FWPD, ZI, MI, SVDI and kNNI at significance level $\alpha = 0.05$. Here, the null hypothesis (H_0) is that there is no significant difference between the two comparing methods. We have used both NMI [36] and ARI [37] (reported in Table I) for this experiment. When NMI is used for comparison, the performance of the proposed method is comparable to HAC-FWPD and significantly better than all other comparing methods. Similar to NMI, when ARI is used we found

that the performance of the proposed method is comparable to HAC-FWPD and significantly better than all other comparing methods.

V. Conclusion

In this article, we have proposed two innovative methods for dealing with missing values so that data with missing attributes can be clustered effectively. Many real-life applications suffers from missing values. For example, it is common for gene expression data generated by DNA-Microarray technology to have missing values due to factors like as experimental error and hybridization failure. Almost in every application of microarray data, we have very limited number of instances. Moreover, repetitions of the experiment do not guarantee that there will not be any missing values. Typically, one ends up with different sets of missing values. For knowledge discovery from such data sets, for example, finding sub-types of the disease, we cannot discard samples with missing values. In such a case, we can use the proposed method to generate latent space representation of all instances and then the clustering can be done in the latent space. Finally, those clusters can be projected back to the original data space. We note here that for data sets with $p >> n$ an autoencoder with a bottleneck layer might be more useful.

We have tested the proposed algorithm on twelve challenging datasets (taken from University of California at Irvine (UCI) repository) and compared the results with five state-of-the-art techniques in terms of NMI and ARI. Of the 12 datasets, our first method performs the best for 9 datasets. Experimentally we also demonstrate that by changing the architecture the performance of the proposed method is marginally changed. We have repeated all experiment 5 times for 10 initial conditions of the autoencoder and 10 initial partition matrices for FCM and have taken the average NMI and ARI to show our performance. Here, for all datasets, we take the information of the first hidden layer for clustering.

An autoencoder tries to learn an identity map but, as such, it does not care about the structure of input data. Since our objective here is to cluster an incomplete data set, if we can preserve the inter-point distance structure of the input data in the latent space, we expect to be able to extract the same/similar clusters which are present in the input data. Keeping this in view, we extended our method by adding Sammon's stress as a penalty to the usual objective function of the autoencoder. To reduce the computational overhead, we then simplify the penalty using the prototypes extracted by a SOM. Our experimental results reveal that this method is able to achieve its objective.

For our first methods the size of the augmented training set is $(p+1) \times n$. If dimension of a dataset, i.e., p is large then $(p+1) \times n$ is also large which demand more computational resources for large p. But we cannot reduce the overhead due to large p as any feature may be missing for any data point. However, today's computing hardware, when available, can address this issue. We may reduce the overhead due to the large n using different schemes. First, we can choose a small percentage of random instances to generate the augmented training data. As an example if we use 20% of the samples, then the size of the augmented training data would $n + \frac{p \times n}{5} = n \times (1 + \frac{p}{5})$. Second, we can use some sophisticated method like SOM to quantize the data and use only a few instances from the set of instances that are mapped to each SOM prototype. Note that, this extension using SOM is different from the one that we have studied in this paper. In the future, we like to explore some of these ideas.

References

[1] R. J. Hathaway and J. C. Bezdek, "Fuzzy c-means clustering of incomplete data," *IEEE Trans. Syst., Man, Cybern., Part B (Cybern.)*, vol. 31, no. 5, pp. 735–744, Oct. 2001.

[2] P. J. García-Laencina, J.-L. Sancho-Gómez, and A. R. Figueiras-Vidal, "Pattern classification with missing data: A review," *Neural Comput. Appl.*, vol. 19, no. 2, pp. 263–282, 2010.

[3] L. N. Nguyen and W. T. Scherer, "Imputation techniques to account for missing data in support of intelligent transportation systems applications," Center for Transportation Studies, University of Virginia, Charlottesville, Res. Rep. UVACTS-13-0-78, May 2003.

[4] K. Lakshminarayan, S. A. Harp, and T. Samad, "Imputation of missing data in industrial databases," *Appl. Intell.*, vol. 11, no. 3, pp. 259–275, 1999.

[5] C. Ji and A. Elwalid, "Measurement-based network monitoring: Missing data formulation and scalability analysis," in *Proc. IEEE Inf. Theory Int. Symp.*, 2000, p. 78.

[6] M. H. Le Gruenwald, "Estimating missing values in related sensor data streams," in *Proc. Int. Conf. Data Sci. Manage. Data*, 2005, pp. 83–94.

[7] H. S. Mohammed, N. Stepenosky, and R. Polikar, "An ensemble technique to handle missing data from sensors," in *Proc. IEEE Sensors Appl. Symp.*, 2006, pp. 101–105.

[8] M. Cooke, P. D. Green, and M. Crawford, "Handling missing data in speech recognition," in *Proc. Int. Conf. Spoken Lang. Process.*, 1994, pp. 1555–1558.

[9] S. Parveen and P. Green, "Speech enhancement with missing data techniques using recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2004, vol. 1, pp. I–733.

[10] G. DiCesare, "Imputation, estimation and missing data in finance," Ph.D. dissertation, Waterloo, Ontario, Canada, 2006.

[11] P. Kofman and I. G. Sharpe, "Using multiple imputation in the analysis of incomplete observations in finance," *J. Financial Econometrics*, vol. 1, no. 2, pp. 216–249, 2003.

[12] P. Liu, E. El-Darzi, L. Lei, C. Vasilakis, P. Chountas, and W. Huang, "An analysis of missing data treatment methods and their application to health care dataset," in *Proc. Int. Conf. Adv. Data Mining Appl.*, 2005, pp. 583–590.

[13] M. K. Markey and A. Patel, "Impact of missing data in training artificial neural networks for computer-aided diagnosis," in *Proc. Mach. Learn. Appl. Int. Conf.*, 2004, pp. 351–354.

[14] M. A. Proschan *et al.*, "Sensitivity analysis using an imputation method for missing binary data in clinical trials," *J. Statist. Planning Inference*, vol. 96, no. 1, pp. 155–165, 2001.

[15] J. M. Jerez, I. Molina, J. L. Subirats, and L. Franco, "Missing data imputation in breast cancer prognosis," *BioMed*, vol. 6, pp. 323–328, 2006.

[16] J. Liu, C. Wang, J. Gao, and J. Han, "Multi-view clustering via joint nonnegative matrix factorization," in *Proc. SIAM Int. Conf. Data Mining*, 2013, pp. 252–260.

[17] S.-Y. Zhi and H. Zhou, "Partial multi-view clustering," in *Proc. AAAI Conf. Artif. Intell.*, 2014, pp. 1968–1974.

[18] W. Shao, L. He, and S. Y. Philip, "Multiple incomplete views clustering via weighted nonnegative matrix factorization with $l_{2,1}$ regularization," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, 2015, pp. 318–334.

[19] H. Zhao, H. Liu, and Y. Fu, "Incomplete multi-modal visual data grouping," in *Proc. Int. Joint Conf. Artif. Intell.*, 2016, pp. 2392–2398.

[20] Q. Yin, S. Wu, and L. Wang, "Unified subspace learning for incomplete and unlabeled multi-view data," *Pattern Recognit.*, vol. 67, pp. 313–327, 2017.

[21] L. Zhao, Z. Chen, Y. Yang, Z. J. Wang, and V. C. Leung, "Incomplete multi-view clustering via deep semantic mapping," *Neurocomputing*, vol. 275, pp. 1053–1062, 2018.

[22] L. Zhang, Y. Zhao, Z. Zhu, D. Shen, and S. Ji, "Multi-view missing data completion," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 7, pp. 1296–1309, Jul. 2018.

[23] D. Li, H. Gu, and L. Zhang, "A fuzzy c-means clustering algorithm based on nearest-neighbor intervals for incomplete data," *Expert Syst. Appl.*, vol. 37, no. 10, pp. 6942–6947, 2010.

[24] D. Li, H. Gu, and L. Zhang, "A hybrid genetic algorithm–fuzzy c-means approach for incomplete data clustering based on nearest-neighbor intervals," *Soft Comput.*, vol. 17, no. 10, pp. 1787–1796, 2013.
[25] B. Wang, L. Zhang, L. Zhang, Z. Bing, and X. Xu, "Missing data imputation by nearest-neighbor trained bp for fuzzy clustering," *J. Inf. Comput. Sci.*, vol. 11, no. 15, pp. 5367–5375, 2014.
[26] J. Li, S. Song, Y. Zhang, and Z. Zhou, "Robust K-median and K-means clustering algorithms for incomplete data," *Math. Problems Eng.*, vol. 2016, 2016, Art. no. 4321928.
[27] J. Li, S. Song, Y. Zhang, and K. Li, "A robust fuzzy c-means clustering algorithm for incomplete data," in *Proc. Intell. Comput., Netw. Control, Eng. Appl.*, 2017, pp. 3–12.
[28] S. Datta, S. Bhattacharjee, and S. Das, "Clustering with missing features: A penalized dissimilarity measure based approach," 2016, *arXiv:1604.06602*.
[29] K. Honda and H. Ichihashi, "Linear fuzzy clustering techniques with missing values and their application to local principal component analysis," *IEEE Trans. Fuzzy Syst.*, vol. 12, no. 2, pp. 183–193, Apr. 2004.
[30] A. Krizhevsky and G. E. Hinton, "Using very deep autoencoders for content-based image retrieval," in *Proc. Eur. Symp. Artif. Neural Netw.*, 2011.
[31] S. Lange and M. A. Riedmiller, "Deep auto-encoder neural networks in reinforcement learning," in *Proc. Int. Joint Conf. Neural Netw.*, 2010, pp. 1–8.
[32] L. Deng, M. L. Seltzer, D. Yu, A. Acero, A.-R. Mohamed, and G. Hinton, "Binary coding of speech spectrograms using a deep auto-encoder," in *Proc. 11th Annu. Conf. Int. Speech Commun. Assoc.*, 2010, t pp. 1692–1695.
[33] M. Sun, X. Zhang, H. Van Hamme, and T. F. Zheng, "Unseen noise estimation using separable deep auto encoder for speech enhancement," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 24, no. 1, pp. 93–104, Jan. 2016.
[34] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 1096–1103.
[35] X. Yu, H. Li, Z. Zhang, and C. Gan, "The optimally designed variational autoencoder networks for clustering and recovery of incomplete multimedia data," *Sensors*, vol. 19, no. 4, p. 809, 2019.
[36] A. Strehl and J. Ghosh, "Cluster ensembles—A knowledge reuse framework for combining multiple partitions," *J. Mach. Learn. Res.*, vol. 3, no. Dec, pp. 583–617, 2002.
[37] K. Y. Yeung and W. L. Ruzzo, "An empirical study on principal component analysis for clustering gene expression data," *Bioinformatics*, vol. 17, no. 9, pp. 763–774, 2001.
[38] S. J. Choudhury and N. R. Pal, "Imputation of missing data with neural networks for classification," *Knowl. Based Syst.*, vol. 182, 2019, Art. no. 104838.
[39] J. W. Sammon, "A nonlinear mapping for data structure analysis," *IEEE Trans. Comput.*, vol. 100, no. 5, pp. 401–409, May 1969.
[40] R. J. Little and D. B. Rubin, *Statistical Analysis with Missing Data*. Hoboken, NJ, USA: Wiley, 2014.
[41] J. L. Schafer, *Analysis of Incomplete Multivariate Data*. Boca Raton, FL, USA: CRC Press, 1997.
[42] P. D. Allison, "Missing data: Sage university papers series on quantitative applications in the social sciences (07–136)," Thousand Oaks, CA, USA, 2001.
[43] G. Kalton, "Compensating for missing survey data," Survey Research Center, Institute for Social Research, The University of Michigan, ISR Code No. 9016, 1983.
[44] J. K. Dixon, "Pattern recognition with partly missing data," *IEEE Trans. Syst., Man, Cybern.*, vol. 9, no. 10, pp. 617–621, Oct. 1979.
[45] R. Morin and B. Raeside, "A reappraisal of distance-weighted *k*-nearest neighbor classification for pattern recognition with missing data," *IEEE Trans. Syst., Man, Cybern.*, vol. 11, no. 3, pp. 241–243, Mar. 1981.
[46] T. Samad and S. A. Harp, "Self–organization with partial data," *Netw. Comput. Neural Syst.*, vol. 3, no. 2, pp. 205–212, 1992.
[47] F. Fessant and S. Midenet, "Self-organising map for data imputation and correction in surveys," *Neural Comput. Appl.*, vol. 10, no. 4, pp. 300–310, 2002.
[48] L. K. Westin, *Missing Data and the Preprocessing Perceptron*, Ph.D. dissertation, Dept. Comput. Sci., Umeå Univ., SE-90187 Umeå, Sweden, 2004.
[49] R. Nowicki, "Rough neuro-fuzzy structures for classification with missing data," *IEEE Trans. Syst., Man, Cybern., Part B (Cybern.)*, vol. 39, no. 6, pp. 1334–1347, Dec. 2009.
[50] C. Gautam and V. Ravi, "Counter propagation auto-associative neural network based data imputation," *Inf. Sci.*, vol. 325, pp. 288–299, 2015.
[51] E.-L. Silva-Ramírez, R. Pino-Mejías, and M. López-Coello, "Single imputation with multilayer perceptron and multiple imputation combining multilayer perceptron and k-nearest neighbours for monotone patterns," *Appl. Soft Comput.*, vol. 29, pp. 65–74, 2015.
[52] C. Gautam and V. Ravi, "Data imputation via evolutionary computation, clustering and a neural network," *Neurocomputing*, vol. 156, pp. 134–142, 2015.
[53] E.-L. Silva-Ramírez, R. Pino-Mejías, M. López-Coello, and M.-D. Cubiles-de-la Vega, "Missing value imputation on missing completely at random data using multilayer perceptrons," *Neural Netw.*, vol. 24, no. 1, pp. 121–129, 2011.
[54] Q. Ma, Y. Gu, W.-C. Lee, and G. Yu, "Order-sensitive imputation for clustered missing values," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 1, pp. 166–180, Jan. 2019.
[55] Z.-G. Liu, Q. Pan, G. Mercier, and J. Dezert, "A new incomplete pattern classification method based on evidential reasoning," *IEEE Trans. Cybern.*, vol. 45, no. 4, pp. 635–646, Apr. 2015.
[56] J. Dai, H. Hu, Q. Hu, W. Huang, N. Zheng, and L. Liu, "Locally linear approximation approach for incomplete data," *IEEE Trans. Cybern.*, vol. 48, no. 6, pp. 1720–1732, Jun. 2018.
[57] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.
[58] A. G. Di Nuovo, "Missing data analysis with fuzzy c-means: A study of its application in a psychological scenario," *Expert Syst. Appl.*, vol. 38, no. 6, pp. 6793–6797, 2011.
[59] J. C. Bezdek, R. Ehrlich, and W. Full, "FCM: The fuzzy c-means clustering algorithm," *Comput. Geosci.*, vol. 10, no. 2/3, pp. 191–203, 1984.
[60] D. Bhowmik, "Domain adaptation by preserving topology," Master's thesis, Electron. Commun. Sci. Unit, Indian Statistical Inst., Kolkata, India, 2018.
[61] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," La Jolla Inst. Cogn. Sci., California Univ. San Diego, San Diego, CA, USA, Tech. Rep. ICS 8506, 1985.
[62] S. Kumar, *Neural Networks: A Classroom Approach*. New York, NY, USA: McGraw-Hill, 2004.
[63] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.*, vol. 43, no. 1, pp. 59–69, 1982.
[64] "University of california at irvine (UCI) repository of machine learning databases," 2005. [Online]. Available: ftp:/ftp.ics.uci.edu/pub/machine-learning-databases
[65] J. Bezdek and N. Pal, "Some new indexes of cluster validity," *IEEE Trans. Syst., Man, Cybern., Part B (Cybern.)*, vol. 28, no. 3, pp. 301–315, Jun. 1998.
[66] N. R. Pal and J. C. Bezdek, "On cluster validity for the fuzzy c-means model," *IEEE Trans. Fuzzy Syst.*, vol. 3, no. 3, pp. 370–379, Aug. 1995.
[67] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, no. Dec, pp. 3371–3408, 2010.

Suvra Jyoti Choudhury (S'18) received the B.Tech. degree in computer science and engineering from the West Bengal University of Technology, Kolkata, India, and the M.Tech. degree in information technology from the Indian Institute of Engineering Science and Technology, Shibpur, India, in 2009 and 2014, respectively, and is currently working toward the Ph.D. degree from Indian Statistical Institute, Kolkata, India. His current research interests include image processing, artificial neural networks, and machine learning.

Nikhil Ranjan Pal (M'91–SM'00–F'05) is a Professor in the Electronics and Communication Sciences Unit and the Head of the Center for Artificial Intelligence and Machine Learning of the Indian Statistical Institute. He was an Editor-in-Chief of the IEEE TRANSACTIONS ON FUZZY SYSTEMS for the period 2005–2010. He is the recipient of the 2015 IEEE Computational Intelligence Society (CIS) Fuzzy Systems Pioneer Award. At present, he is serving as the President of the IEEE CIS (2018–2019). He is a Fellow of the National Academy of Sciences, India, Indian National Academy of Engineering, Indian National Science Academy, International Fuzzy Systems Association (IFSA) and The World Academy of Sciences.