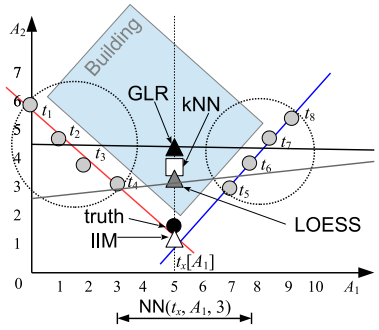
2019 IEEE 35th International Conference on Data Engineering (ICDE)

Learning Individual Models for Imputation

Aoqian Zhang, Shaoxu Song, Yu Sun, Jianmin Wang
BNRist, Tsinghua University, Beijing, China
{zaq13, sxsong, sy17, jimwang}@tsinghua.edu.cn

***Abstract*—Missing numerical values are prevalent, e.g., owing to unreliable sensor reading, collection and transmission among heterogeneous sources. Unlike categorized data imputation over a limited domain, the numerical values suffer from two issues: (1) *sparsity problem*, the incomplete tuple may not have sufficient complete neighbors sharing the same/similar values for imputation, owing to the (almost) infinite domain; (2) *heterogeneity problem*, different tuples may not fit the same (regression) model. In this study, enlightened by the conditional dependencies that hold conditionally over certain tuples rather than the whole relation, we propose to learn a regression model individually for each complete tuple together with its neighbors. Our IIM, *Imputation via Individual Models*, thus no longer relies on sharing similar values among the k complete neighbors for imputation, but utilizes their regression results by the aforesaid learned individual (not necessary the same) models. Remarkably, we show that some existing methods are indeed special cases of our IIM, under the extreme settings of the number ℓ of learning neighbors considered in individual learning. In this sense, a proper number ℓ of neighbors is essential to learn the individual models (avoid over-fitting or under-fitting). We propose to adaptively learn individual models over various number ℓ of neighbors for different complete tuples. By devising efficient incremental computation, the time complexity of learning a model *reduces from linear to constant*. Experiments on real data demonstrate that our IIM with adaptive learning achieves higher imputation accuracy than the existing approaches.**



Tid	A_1	A_2
t_1	0	5.8
t_2	0.8	4.6
t_3	1.9	3.8
t_4	2.9	3.2
t_5	6.8	3
t_6	7.5	4.1
t_7	8.2	4.8
t_8	9	5.5
t_x	5	–

Fig. 1: Motivation example of two-dimension data, where $t_x[A_2]$ is a missing value with ground truth 1.8. Our IIM learns the individual regression models (blue and red lines) w.r.t. heterogeneous neighbors (t_4 and t_5), instead of a same model (black or gray line) for all neighbors

I. Introduction

Missing values are commonly observed over numerical data [16], for instance, owing to failures of sensor reading devices [17], poorly handling overflow during calculation, mismatching in integrating heterogeneous sources [13], and so on. Simply discarding the incomplete tuples with missing values makes the data even more incomplete.

A. Motivation

We notice that the existing imputation techniques [23], [3], [6] utilizing either complete attributes or complete tuples suffer from two major issues, especially when handling numerical data from various sources. (See examples below.)

1) Sparsity problem: The imputation via finding the closest complete tuple relies on the assumption that there exist neighbors sharing the same values. Unfortunately, owing to the sparsity issue, such an assumption is often not the case in practice, e.g., t_x in Figure 1 does not have any complete tuple sharing the same value. Thereby, the kNN method [3] proposes to aggregate the values of complete neighbors.

Owing to sparsity, there may not exist a complete tuple containing exactly the actual correct value of the incomplete tuple. For this reason, it is also studied to impute a missing value from the regression model [22]. Instead of using a value directly from the complete tuple (often unlikely to be the actual correct value owing to sparsity), the prediction based approach (GLR) [22] assumes tuples sharing regression models. For instance, in Figure 1, t_5 and t_6 have different values, but share the same regression model (blue line).

It is worth noting that even a complete tuple (t_5) is trusted (with no error), its value cannot be directly used as the imputation of the incomplete tuple (t_x), owing to the aforesaid sparsity issue. However, this tuple t_5 can be used to learn a regression model. The incomplete tuple t_x may not directly use the value of t_5, but use the value predicted by regression model of t_5, since neighbors may not share the same value but the regression model.

2) Heterogeneity problem: Since data often describe various facts or are collected from heterogeneous sources, no global semantics may fit the entire data [29]. That is, there may not exist a single regression model that captures the semantics over all the data. Or generally speaking, *one size does not fit all*. For instance, in Figure 1, a single GLR model (black line) cannot fit all the data points in different streets.

To address the heterogeneity problem, instead of assuming the same regression, we argue to learn a fine-grained individual regression model that is only valid locally over a complete tuple and its neighbors. For instance, in Figure 1, the individual regression model (red line) is only valid over t_4 and its neighbors such as t_3. Tuple t_5 in another street could have another regression model (blue line) that is distinct from t_4. The imputation can thus utilize these more accurate individual models, instead of the imprecise global model (GLR) that does

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00023

not fit all the data. The benefit of the imputation by individual models (IIM) would be the clearly higher accuracy than that of GLR with a single (inaccurate) global model, as the results shown in Table V.

Example 1. *Consider a check-in dataset of two dimension in Figure 1 for simplicity (more general, high dimensional data are considered in Section VI of experiments). Tuples $t_1 - t_8$ (denoted by gray dots) represent 8 observations in the streets outside a building. There is another tuple t_x with $t_x[A_1] = 5$ observed but $t_x[A_2]$ missing during transmission (the truth of $t_x[A_2]$ is denoted by the black dot).*

The nearest neighbor based imputation finds k (say $k = 3$) tuples that are most similar to t_x on the complete attribute A_1, i.e., t_4, t_5, t_6. The mean value of three tuples on A_2 is then considered as the imputation of t_x (kNN, white square). Unfortunately, since no tuple is sufficiently close to the truth of t_x (owing to sparsity), the imputation is not accurate.

The global linear regression (represented by solid black line) obviously cannot capture the difference between observations $t_1 - t_4$ and $t_5 - t_8$ in two streets. The imputation by the global regression (GLR, black triangle) is not accurate.

The local regression assumes a same regression locally over the neighbors t_4, t_5, t_6 of the incomplete tuple t_x, found on the complete attribute A_1. Again, owing to the heterogeneity issue, t_5, t_6 and t_4 from two streets, respectively, indeed have different regression models. The imputation by the local regression (LOESS, gray triangle) is not accurate either.

The idea of IIM is enlightened by the conditional dependencies [7], which only hold conditionally over certain tuples rather than the whole relation. That is, the constraint does not fit all the data, but only applies to a subset of tuples specified by certain conditions. Analogously, a regression model may not fit all the data, but only applies "conditionally" to the nearby neighbors of a tuple. Thereby, we propose to learn a regression model individually for each complete tuple and its neighbors, instead of a single global regression model that cannot fit all the tuples.

B. Proposal

The Imputation via Individual Models (IIM) proposed in this paper thus has two phases: (1) the *learning phase* learns individually a regression for each complete tuple together with its neighbors, e.g., $f_1, \ldots, f_3$ for $t_1, \ldots, t_3$, respectively, in Figure 2; and (2) the *imputation phase* finds k complete imputation neighbors of the incomplete tuple, and aggregate the regression results produced by the aforesaid learned individual regression models of the k complete neighbors.

For example, t_x could use the regression models of neighbors t_4, t_5 and t_6, and aggregate the results of different regressions as the imputation (IIM, white triangle in Figure 1).

A key issue is how to perform individual learning for each complete tuple. To learn the individual model, it needs to find a number of ℓ learning neighbors that are similar to the tuple. A different number of learning neighbors lead to various learned models. Determining the number ℓ of neighbors for learning is highly non-trivial. For each complete tuple, (1) if the number ℓ is too small, the learned regression model may *overfit* the data; (2) on the other hand, if ℓ is too large (e.g., considering almost all the heterogeneous tuples in the dataset like global regression), it leads to *under-fitting*. (We address the overfitting and under-fitting issues by adaptive learning below.)

TABLE I: Notations

$\mathcal{R}$	schema on m attributes
r	relation of n complete tuples
t_x	incomplete tuple with missing value
A_x	incomplete attribute in t_x, A_m by default for simplicity
$\mathcal{F}$	complete attributes in t_x, $\mathcal{R} \setminus \{A_m\}$ by default
ϕ	parameter of linear regression model
ℓ	number of learning neighbors for learning the individual model of a complete tuple
k	number of imputation neighbors for imputing an incomplete tuple

C. Contribution

Our major contributions in this paper are summarized as:

(1) We propose a novel approach IIM of Imputation via Individual Models (Section III), with learning and imputation phases as aforesaid. The heterogeneity issue is addressed by learning an individual model for each tuple together with its neighbors. IIM does not directly use the values of complete neighbors for imputation (but their models) and thus tackles the sparsity problem.

(2) We prove that some existing approaches are indeed the special cases of IIM under extreme settings (i.e., $\ell = 1$ or $\ell = n$ in Propositions 1 and 2 in Section IV). It does not only illustrate the rationale of our proposal, but also motivate us to adaptively determine a proper ℓ (in between the extreme 1 and n) for each tuple to avoid over-fitting or under-fitting.

(3) We adaptively learn the individual model for each complete tuple over a distinct number ℓ of learning neighbors (Section V). By introducing a validation step, we determine a proper number ℓ and the corresponding learned model for each complete tuple, which can impute most accurately the other complete tuples (considered as validation set). Experiments show that the adaptively learned individual models indeed lead to better imputation results. Efficient incremental computation is devised for adaptive learning, which reduces the time complexity of learning a model from linear to constant.

(4) We conduct extensive experiments over real datasets (Section VI). The results demonstrate that our IIM has significantly better performance than the state-of-the-art imputation methods. Remarkably, we show in [1] that the proposed imputation indeed improves the accuracy of classification application over the data with *real-world missing values*.

Table I lists the frequently used notations.

II. Preliminary and Related Work

In this section, we introduce preliminaries and categorize major imputation approaches into two classes in Table II. The key ideas of imputation based on tuple models and

TABLE II: Imputation methods considered in (empirical) comparison

Approach	Model	Property
Mean [15]	Tuple	Global average
kNN [3]	Tuple	Local average
kNNE [14]	Tuple	kNN Ensemble
IFC [26]	Tuple	Cluster average
GMM [31]	Tuple	Cluster average
SVD [30]	Tuple	k most significant eigengenes
ILLS [9]	Tuple	Local regression over tuples
GLR [22]	Attribute	Global regression
LOESS [11]	Attribute	Local regression
BLR [28]	Attribute	Bayesian linear regression
ERACER [24]	Attribute	Neighbor regression
PMM [19]	Attribute	Predictive mean matching
XGB [10]	Attribute	Xgboost, tree boosting system

attribute models are presented in Figure 2. We discuss that each category of existing techniques suffers from either the heterogeneity or the sparsity problem. It motivates us to devise the novel imputation via individual models in Section III.

Consider a relation r of n tuples $r = \{t_1, t_2, \ldots, t_n\}$, with schema $\mathcal{R} = \{A_1, A_2, \ldots A_m\}$ on m attributes. We denote $t_i[A_j]$ the value of tuple $t_i \in r$ on attribute $A_j \in \mathcal{R}$.

Let t_x be a tuple over $\mathcal{R}$ with missing value on attribute A_x. We call A_x the incomplete attribute and $\mathcal{F} = \mathcal{R} \setminus \{A_x\}$ the complete attributes. (For simplicity, we consider A_m as the incomplete attribute by default. Missing values on other attributes could be addressed similarly. Multiple incomplete attributes in a tuple could be addressed one by one.)

A. Imputation based on Tuple Models

1) Nearest Neighbor Model kNN: To impute the missing numerical values, a natural idea is to retrieve similar complete instances from r for imputation, known as the *k-nearest-neighbor* approach, kNN [3], [6].

Let $\mathsf{NN}(t_x, \mathcal{F}, k)$ be k nearest neighbors of t_x on attributes $\mathcal{F}$ from r, e.g., with the smallest Euclidean distance [4]

$$d_{x,i} = \sqrt{\frac{\sum_{A \in \mathcal{F}} (t_x[A] - t_i[A])^2}{|\mathcal{F}|}} \tag{1}$$

where $d_{x,i}$ denotes the distance between tuple t_x and t_i on complete attributes $\mathcal{F}$.

The kNN imputation is in two steps: (1) find k nearest neighbors $T_x = \mathsf{NN}(t_x, \mathcal{F}, k)$, and (2) use the A_m values of neighbors for imputation, e.g., by arithmetic mean

$$t'_x[A_m] = \frac{\sum_{t_j \in T_x} t_j[A_m]}{k}. \tag{2}$$

2) Variations of Tuple Models: The first variation is on the neighbors in step (1) of the kNN imputation. kNNE [14] finds different groups of k neighbors by computing distances on various subsets of features and then combine the imputation results from these different groups. Instead of k neighbors, the Mean method [15] simply identifies all the tuples (as T_x) for aggregation in the following step. Clustering is also employed to identify the neighbors for imputation, e.g., IFC [26] considering fuzzy k-means [20] or GMM [31] using the Gaussian mixture model. Moreover, instead of searching existing data as neighbors, the SVD approach [30] finds a set of mutually orthogonal expression patterns (so-called eigenvectors) as T_x for aggregation imputation.

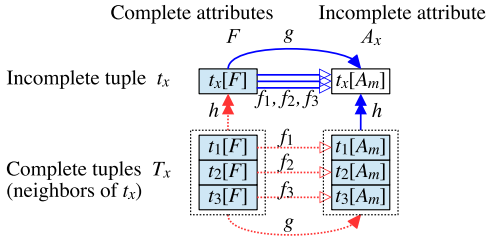


Fig. 2: Learning (dashed arrows) models over complete data and imputing (solid arrows) the missing value $t_x[A_m]$ by tuple model h, attribute model g, or individual models $f_1, \ldots, f_3$ w.r.t. $t_1, \ldots, t_3$

The second variation is on the aggregation model in step (2) of the kNN imputation. In addition to Formula 2, more advanced aggregation considers the distances of neighbors as aggregation weights [4]. Furthermore, instead of the model of aggregating $t_j[A_m]$ over $t_j \in T_x$, ILLS [9] learns a model h for predicting t_x values from T_x. In this sense, the arithmetic mean aggregation in Formula 2 is a special h that does not need learning from $T_x[\mathcal{F}]$ and $t_x[\mathcal{F}]$. We call h a tuple model, and this category the tuple model-based imputation.

3) Discussion: The idea of learning over individual tuples and their closest neighbors in our proposal IIM is related to past work kNN [3]. The difference is that to impute the incomplete tuple t_x, kNN uses (aggregates) directly the values of the k-closest neighbors t_i of t_x as the imputation, while our IIM learns individual models for the neighbor tuples t_i by considering their ℓ-closest neighbors t_j, respectively. The values predicted by the learned models are then aggregated as the imputation. The defeat of directly using the values of k-closest neighbors to impute missing values is that owing to sparsity, no sufficient neighbors could be found sharing similar values with incomplete tuple t_x. For instance, t_x in Figure 1 does not have any tuple sharing highly similar values. Alternatively, we learn a model from the tuple and its ℓ-closest neighbors. Tuples may not share the same values but models. For example, t_x in Figure 1 fits the model that is learned from t_4 and its neighbors, and is thus accurately imputed.

B. Imputation based on Attribute Models

1) Linear Regression Model GLR: Rather than capturing relationships to the complete tuples in r, another well-known idea is to explore the relationships between incomplete and complete attributes, e.g., by the linear regression model [23].

Let $\mathsf{LR}(\mathcal{F}, A_m, \mathcal{R})$ denote the linear regression model from complete attributes $\mathcal{F}$ to incomplete attribute A_m, having

$$\begin{aligned} t[A_m] &= \phi[C]1 + \phi[A_1]t[A_1] + \ldots + \phi[A_{m-1}]t[A_{m-1}] + \varepsilon \\ &= (1, t[\mathcal{F}])\phi + \varepsilon \end{aligned} \tag{3}$$

where t is a tuple over $\mathcal{R}$, $\phi = \{\phi[C], \phi[A_1], \ldots, \phi[A_{m-1}]\}^\top$ is the parameter of linear regression ($\phi[C]$ denotes the constant term), and ε is the error term.

The imputation is thus in two steps: (1) learn parameter ϕ from relation r of complete tuples (e.g., by Ordinary Least Square or Ridge Regression [27], see more details in Section III-A), and (2) perform the imputation referring to the learned linear regression model,

$$t'_x[A_m] = (1, t_x[\mathcal{F}])\phi. \tag{4}$$

Since the linear regression is declared on all tuples over $\mathcal{R}$, we call this *global linear regression* method, GLR.

2) Variations on Attribute Models: Similar to the idea of aggregating only kNN tuples [3] rather than Mean [15] of all tuples in Section II-A, LOESS [11] learns a local regression over the neighbors $\mathsf{NN}(t_x, \mathcal{F}, k)$ of t_x, instead of the global regression over all tuples. Statistical analysis could be further employed to linear regression, e.g., Bayesian linear regression BLR in the context of Bayesian inference. (We use the MICE [8] implementation mice.norm in R in experiments.)

The attribute models can cooperate with the tuple models. The ERACER approach [24] further studies the regression model over neighbors, i.e., combining both g and h in Figure 2. For instance, the temperature of a sensor is related to its humidity (g), as well as its neighbors' temperature and humidity (h). The predictive mean matching PMM [19] does not directly use the value $t'_x[A_m]$ predicated by linear regression as the imputation. Instead, it finds neighbors whose predications also by the same linear regression are most similar to the predicated value $t'_x[A_m]$. A randomly selected original value $t_j[A_m]$ of the identified neighbors t_j is returned as the imputation. The widely used XGboost [10] (XGB) algorithm learns a set of classification and regression trees and ensembles the results. (We use the MICE [8] implementation mice.pmm and library 'xgboost' in R in the experiments.)

3) Discussion: Owing to the heterogeneity problem, assuming the same regression either globally, locally or randomly (for xgboost) [11] for different tuples could be indefensible.

III. Imputation via Individual Learning

As illustrated in Figure 2, the Imputation via Individual Models (IIM) addresses the heterogeneity and sparsity problems in two aspects, respectively. (1) The learning phase in Section III-A learns a linear regression model individually for each tuple (together with its neighbors, e.g., models $f_1, \ldots, f_3$ in Figure 2), instead of assuming the same regression for different tuples (with heterogeneity). This is enlightened by the conditional dependencies that hold conditionally over certain tuples [7]. (2) The imputation phase in Section III-B aggregates the regression results of multiple individual regression models suggested by different neighbors, rather than relying the neighbors to have similar values (suffering sparsity).

A. Learning Phase

The *learning phase* learns the parameter ϕ_i of the linear regression model (in Formula 3) individually for each tuple $t_i \in r$. The learned individual regression models are then utilized in the imputation in Section III-B.

Algorithm 1: Learning($r, \ell, \mathcal{F}, A_m$)

Input: relation r of complete tuples, number ℓ of learning neighbors, complete attributes $\mathcal{F}$, incomplete attribute A_m

Output: Φ the set of regression parameters ϕ_i learned for all tuples t_i in r

1 **for** *each* $t_i \in r$ **do**
2 $\quad T_i \leftarrow \mathsf{NN}(t_i, \mathcal{F}, \ell)$;
3 $\quad \phi_i \leftarrow \mathsf{LR}(\mathcal{F}, A_m, T_i)$;
4 **return** Φ

Algorithm 1 presents the procedure of individual learning over r for the regression from $\mathcal{F}$ to A_m. For each $t_i \in r$, we consider a set T_i of nearest neighbors i.e., $\mathsf{NN}(t_i, \mathcal{F}, \ell)$ in Line 2, a.k.a. learning neighbors. They are obtained in the same way of obtaining k nearest neighbors in the kNN approach, $\mathsf{NN}(t_x, \mathcal{F}, k)$, as introduced in Section II-A1. That is, return the tuples with the smallest Euclidean distance on attributes $\mathcal{F}$ [4]. In case of sparsity, the returned neighbors may not share similar values with the incomplete tuple, and thus the kNN approach directly aggregating the values of nearest neighbors is not accurate. To deal with sparsity, we propose to learn regression models over the nearest neighbors, and use the learned models to predict the missing value.

Let ℓ be the number of t_i's neighbors considered in individual learning, namely the number of learning neighbors. As stated in Section I-A, the number ℓ should be sufficiently large to avoid overfitting, but not too large owing to heterogeneity. A straightforward idea is to simply consider a fixed number ℓ for all the tuples in r (see Section VI-C2 for empirical results on considering various fixed ℓ). More advanced adaptive learning considering distinct number of learning neighbors for various tuples in r is devised in Section V.

1) Learning Regression Parameter: Given a set of tuples, $T_i = \{t_1, t_2, \ldots, t_\ell\} \subseteq r$, we employ Ridge Regression [27] to learn the parameter ϕ_i for the regression over T_i,

$$\phi_i = (\boldsymbol{X}^\top \boldsymbol{X} + \alpha \boldsymbol{E})^{-1} \boldsymbol{X}^\top \boldsymbol{Y} \tag{5}$$

where α is regularization parameter, $\boldsymbol{E}$ is identity matrix [25], $\phi_i = \{\phi_i[C], \phi_i[A_1], \ldots, \phi_i[A_{m-1}]\}^\top$,

$$\boldsymbol{Y} = \begin{pmatrix} t_1[A_m] \\ t_2[A_m] \\ \vdots \\ t_\ell[A_m] \end{pmatrix}, \tag{6}$$

$$\boldsymbol{X} = \begin{pmatrix} 1 & t_1[A_1] & t_1[A_2] & \ldots & t_1[A_{m-1}] \\ 1 & t_2[A_1] & t_2[A_2] & \ldots & t_2[A_{m-1}] \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_\ell[A_1] & t_\ell[A_2] & \ldots & t_\ell[A_{m-1}] \end{pmatrix}. \tag{7}$$

$\mathsf{LR}(\mathcal{F}, A_m, T_i)$ in Line 3 computes the parameter ϕ_i over T_i. It returns Φ the set of parameters ϕ_i for all tuples t_i.

Algorithm 2: Imputation(t_x, k, Φ)

Input: t_x the tuple with missing value on attribute A_m, k the number of imputation neighbors, Φ individual regression parameters for all tuples in r

Output: imputation $t'_x[A_m]$

1 $T_x \leftarrow \mathsf{NN}(t_x, \mathcal{F}, k)$;
2 **for** *each* $t_j \in T_x$ **do**
3 $\quad t_x^j[A_m] \leftarrow \mathsf{Candidate}(\phi_j, t_x[\mathcal{F}])$;
4 $t'_x[A_m] \leftarrow \mathsf{Combine}(\{t_x^j[A_m] \mid t_j \in T_x\})$;
5 **return** $t'_x[A_m]$

Example 2. *Consider relation r in Figure 1. Let $\ell = 4$. According to Algorithm 1, we learn the individual regression for each tuple together with its neighbors in r. For t_1, we have $T_1 = \mathsf{NN}(t_1, \{A_1\}, 4) = \{t_1, t_2, t_3, t_4\}$. The regression is learned from T_1 with parameter $\phi_1 = \{5.56, -0.87\}^\top$. Similar computation applies to other tuples in r, having*

$$\Phi = \begin{pmatrix} \phi_1 & \phi_2 & \dots & \phi_8 \end{pmatrix} = \begin{pmatrix} 5.56 & 5.56 & \dots & -4.36 \\ -0.87 & -0.87 & \dots & 1.11 \end{pmatrix}.$$

2) Handling Single Neighbor: As mentioned in the introduction, a small number ℓ will lead to overfitting. When $\ell = 1$, the nearest neighbor returns only one tuple, i.e., $T_i = \{t_i\}$ which has the smallest distance 0 referring to Formula 11. In this case, it is not sufficient to learn a proper regression model. Hence, we directly set $\phi_i[C] = t_i[A_m]$ and $\phi_i[A_1] = \phi_i[A_2] = \dots \phi_i[A_{m-1}] = 0$.

3) Learning Complexity: Line 2 in Algorithm 1 takes $O(mn)$ time to compute distances of all tuples to t_i, and $O(\ell n)$ to find the ℓ nearest tuples (advanced indexing and searching techniques could be applied, which is not the focus of this study). Referring to Formula 5, Line 3 computes ϕ_i with cost $O(m^2\ell + m^3)$. Thereby, the time complexity of Algorithm 1 is $O(mn^2 + \ell n^2 + m^2\ell n + m^3 n)$.

B. Imputation Phase

The *imputation phase* utilizes the individual regression models of t_x's neighbors from r to compute the imputation candidates. Intuitively, in order to enhance the reliability, rather than only one neighbor, we consider the regressions of k imputation neighbors (see Section VI-C1 for an evaluation on varying the number of imputation neighbors k). These k imputation candidates are then aggregated as the final imputation of t_x.

Algorithm 2 presents major steps of the imputation phase:

(S1) Imputation neighbors. Line 1 finds a set T_x of k nearest neighbors of incomplete tuple t_x from relation r on complete attributes $\mathcal{F}$, i.e., $\mathsf{NN}(t_x, \mathcal{F}, k)$ as imputation neighbors.

(S2) Imputation candidates. Line 3 computes a possible imputation $t_x^j[A_m]$ by using the regression of t_x's neighbor t_j with parameter ϕ_j.

(S3) Combination. Line 4 aggregates the candidates suggested by the regressions of all the t_x's neighbors in T_x to form the final imputation $t'_x[A_m]$.

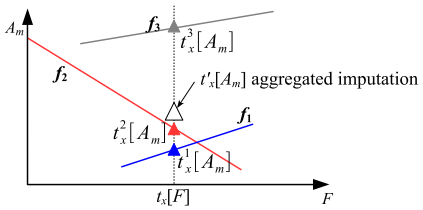


Fig. 3: Intuition example of combining imputation candidates

1) Find imputation neighbors for t_x on complete attributes: This step is the same as step (1) of kNN imputation, i.e., find k nearest neighbors $T_x = \mathsf{NN}(t_x, \mathcal{F}, k)$. However, such neighbors are utilized in a different way. While the kNN imputation directly aggregates the values on attribute A_m of neighbors, e.g., in Formula 2, our proposal considers the individual regression models w.r.t. these neighbors.

2) Imputation via individual regression of each neighbor: For each neighbor $t_j \in \mathsf{NN}(t_x, \mathcal{F}, k)$, we consider the individual regression with parameter ϕ_j learned in the learning phase by Formula 5.

Let t_x^j denote the imputation candidate suggested by the regression of the neighbor t_j. Referring to Formula 3, we have

$$t_x^j[A_m] = (1, t_x[\mathcal{F}])\phi_j + \varepsilon_j, \tag{8}$$

where ε_j is the error term of the regression w.r.t. t_j. It is common to omit the error term ε_j [28] and thus the imputation candidate of the neighbor t_j is

$$t_x^j[A_m] = (1, t_x[\mathcal{F}])\phi_j \tag{9}$$

3) Aggregating individual imputation candidates: In the imputation phase, the tuple t_x with missing values finds complete tuples t_i as its neighbors, and proposes to utilize the aforesaid individually learned models of t_i. Again, owing to heterogeneity (the argument to learn individualized models), not all the neighbors t_i may share the same models with t_x, i.e., not all the neighbors t_i would provide a model leading to the correct value for imputing t_x. Arbitrarily selecting one t_i may lead to the wrong imputation. A neighbor t_i with closer distance to t_x on the complete attribute $\mathcal{F}$ does not denote that its model applies to t_x either. Thereby, we propose a weighted aggregation of the imputation candidates provided by the models of different neighbors t_i, where the candidate values vote for each other.

The aggregated imputation result is defined by

$$t'_x[A_m] = \sum_{t_j \in T_x} t_x^j[A_m] \cdot w_{xj}, \tag{10}$$

where $t_x^j[A_m]$ is the imputation candidate suggested by the imputation neighbor $t_j \in \mathsf{NN}(t_x, \mathcal{F}, k)$, and w_{xj} is the weight of candidate $t_x^j[A_m]$ in aggregation.

Intuitively, we propose to let the candidate values $t_x^i[A_m]$ (provided by the models from different neighbor tuples t_i) vote for each other, via a weighted aggregation function. Similar to the idea of majority voting, those candidate values close with each other are more likely to be the imputation and may

assign higher weights in aggregation, while outliers could be largely ignored with lower aggregation weights. For instance, in Figure 3, the candidates $t_x^1[A_m]$ and $t_x^2[A_m]$ suggested by models f_1 and f_2, respectively, are close and agree with each other. In contrast, the other candidate $t_x^3[A_m]$ by f_3 is outlying (due to heterogeneity), and would be largely ignored with lower aggregation weights.

In this sense, we consider the distances of a candidate $t_x^i[A_m]$ to the other candidates,

$$c_{xi} = \sum_{j=1}^{k} \left| t_x^i[A_m] - t_x^j[A_m] \right|. \tag{11}$$

Following the intuition that candidates close to other (i.e., having smaller c_{xi}) should assign larger weight, we define

$$w_{xi} = \frac{c_{xi}^{-1}}{\sum_{j=1}^{k} c_{xj}^{-1}}, \tag{12}$$

having $\sum_{j=1}^{k} w_{xj} = 1$.

Example 3. *Let $k = 3, \ell = 4$. The imputation starts from the parameter Φ learned in Example 2. Algorithm 2 performs in three steps: (1) Find imputation neighbors for the incomplete tuple t_x, having $T_x = \mathsf{NN}(t_x, \{A_1\}, 3) = \{t_5, t_4, t_6\}$ (2) Compute the imputation candidate via the individual regression of each neighbor. For t_5, referring to the regression model $\mathsf{LR}(\{A_1\}, A_2, T_5)$ with parameter $\phi_5 = (-4.36, 1.11)^\top$, the imputation candidate is computed by $t_x^5[A_2] = (1, 5)(-4.36, 1.11)^\top = 1.19$. Similar computation applies to neighbors t_4 and t_6, having $t_x^4[A_2] = (1, 5)(5.56, -0.87)^\top = 1.21, t_x^6[A_2] = (1, 5)(-4.36, 1.11)^\top = 1.19$. (3) Aggregating the aforesaid imputation candidates. Following Formula 11, we can compute the distance for each imputation candidates as $c_{x5} = c_{x6} = 0.02, c_{x4} = 0.04$. Thus the aggregated imputation by Formula 10 is $t'_x[A_2] = 1.19 * \frac{50}{125} + 1.21 * \frac{25}{125} + 1.19 * \frac{50}{125} = 1.194$.*

4) Imputation Complexity: Similar to the analysis in Section III-A3, Line 1 in Algorithm 2 searches the k nearest neighbors with cost $O(mn + kn)$. The imputation candidates w.r.t. k imputation neighbors are then computed and combined in Lines 3 and 4 with cost $O(mk + k^2)$. Thereby, the time complexity of Algorithm 2 is $O(mn + kn)$.

C. Discussion on Overheads and Benefits

Learning over individual tuples and their ℓ neighbors is a bit more expensive than learning a global model over all the n tuples. Referring to [27], the cost of learning a regression model over n tuples is $O(m^2 n + m^3)$, while the cost of learning n individual models for n tuples given their ℓ neighbors is $O((m^2\ell + m^3)n)$. Nevertheless, both complexities are linear w.r.t. the number of tuples n. In particular, all these models (global or individual) could be offline learned over complete tuples, and directly used in online imputing the missing values of various incomplete tuples.

The benefit of the imputation by individual models (IIM) would be the clearly higher accuracy than that of GLR with a single (inaccurate) global model, as shown in Table V.

IV. Subsuming Existing Methods

To illustrate the rationale of the proposed IIM imputation, in this section, we theoretically prove that some existing methods (kNN [3] and GLR [23] introduced in Sections II-A and II-B) are indeed special cases of our IIM under extreme settings (i.e., $\ell = 1$ or $\ell = n$). It further motivates us to adaptively determine a distinct number of learning neighbors ℓ (in between the extreme 1 and n) for each tuple in Section V.

A. Subsuming kNN

First, we show that IIM subsumes kNN by considering only one learning neighbor in individual learning, i.e., $\ell = 1$.

Proposition 1 (Subsume kNN)**.** *When we consider a fixed number of learning neighbors $\ell = 1$ and a uniform weight of imputation candidate $w_{xi} = \frac{1}{|T_x|}$, the proposed IIM algorithm is equivalent to the kNN imputation. (See proof in [1].)*

B. Subsuming GLR

Moreover, we prove that IIM subsumes GLR by considering all the tuples in r as the learning neighbors in individual learning, i.e., $\ell = n = |r|$.

Proposition 2 (Subsume GLR)**.** *When we consider a fixed number of learning neighbors $\ell = n = |r|$, the IIM algorithm is equivalent to the GLR imputation. (See proof in [1].)*

V. Adaptive Learning

In the learning phase in Section III-A, a fixed number ℓ of learning neighbors is considered for all the tuples in r in Algorithm 1. There are two issues to concern: (1) how to determine a proper number ℓ of neighbors for learning; and (2) different tuples may prefer a distinct number ℓ of learning neighbors, owing to heterogeneity.

In Section V-A, we consider the various candidate regression models learned under different ℓ for a tuple. The adaptive learning (Algorithm 3) selects a proper ℓ as well as the corresponding model for each tuple. Intuitively, to evaluate whether a model learned under some ℓ is proper, we may consider a set of complete tuples as validation data, and see which learned models can best impute the validation tuples (truth is known in the complete validation tuple).

In Section V-B, to efficiently learn the candidate regression models under various ℓ for a tuple, we devise an incremental computing scheme. Remarkably, it reduces the time complexity of individual learning from linear to constant.

A. Adaptive Learning with Validation

Algorithm 3 presents the procedure of adaptively learning a proper regression model from $\mathcal{F}$ to A_m for each complete tuple $t_i \in r$ under various number ℓ of learning neighbors.

First, Line 2 learns the candidate models under various ℓ for all tuples in r, denoted by $\Phi^{(\ell)}$, by call the Learning

Algorithm 3: Adaptive($r, \mathcal{F}, A_m$)

Input: relation r of complete tuples, complete attributes $\mathcal{F}$, incomplete attribute A_m
Output: Φ the set of regression parameters ϕ_i learned for all tuples t_i in r

1 **for** $\ell \leftarrow 1$ **to** n **do**
2 | $\Phi^{(\ell)} \leftarrow \text{Learning}(r, \ell, \mathcal{F}, A_m)$;
3 **for** *each* $t_j \in r$ **do**
4 | $T_j \leftarrow \mathsf{NN}(t_j, \mathcal{F}, k)$;
5 | **for** *each* $t_i \in T_j$ **do**
6 | | **for** $\ell \leftarrow 1$ **to** n **do**
7 | | | $cost[i][\ell] += \left(t_j[A_m] - (1, t_j[\mathcal{F}])\phi_i^{(\ell)}\right)^2$;
8 **for** $i \leftarrow 1$ **to** n **do**
9 | $\ell_i^* \leftarrow \arg\min_{\ell \in [1,n]} cost[i][\ell]$;
10 | $\phi_i \leftarrow \phi_i^{(\ell_i^*)}$;
11 **return** Φ

Algorithm 1. (Advanced incremental computation is devised among different ℓ in Section V-B.)

We consider the complete tuples in r as the validation set. For each $t_j \in r$ employed as a validation tuple, we assume its $t_j[A_m]$ is missing. The original complete value v of $t_j[A_m]$ is directly used to evaluate how the models from t_i (neighbor of t_j) could accurately impute $t_j[A_m]$.

It is worth noting that the model of tuple t_i learned over a number of ℓ learning neighbors can be applied multiple times to impute various t_j. The $cost[i][\ell]$ in Line 7 in Algorithm 3 denotes the total difference between the truths and the imputations given different validation tuples t_j. A model with smaller $cost[i][\ell]$ means more accurate imputation when applied, and thus is preferred in Line 9. This extra overhead is necessary, since we want to select a proper ℓ that performs well in general for imputing potentially all the nearby tuples t_j.

Example 4. *Consider relation r in Figure 1. Suppose that we have learned candidate models under various ℓ for all the tuples in Line 2 in Algorithm 3. Given $k = 3$, we determine a proper model for each tuple from the candidate models $\Phi^{(\ell)}$.*

Let t_1 be the validation tuple. Line 4 finds kNN of t_1 on the complete attribute A_1, i.e., $T_1 = \{t_2, t_3, t_4\}$. For each tuple in T_1, say t_2, the difference between imputation by various candidate models of t_2 and the truth of $t_1[A_2]$ are recorded,

$$cost[2][1] = (5.8 - (1, 0)(4.35, 0)^\top)^2 = 2.1,$$
$$cost[2][2] = (5.8 - (1, 0)(5.79, -1.49)^\top)^2 = 0.0001,$$
$$\dots$$
$$cost[2][8] = (5.8 - (1, 0)(4.41, -0.01)^\top)^2 = 1.93.$$

Line 7 aggregates such difference costs on all the tuples in r (as validation set) in addition to the aforesaid t_1. We have $\{cost[2][1], cost[2][2], \dots, cost[2][8]\} = \{3.73, 3.67, 0.31, \mathbf{0.09}, 1.47, 2.36, 3.03, 3.65\}$. Finally, $\ell_2^* = 4$ with the minimal $cost[2][4]$ is selected and $\phi_2 = \phi_2^{(4)} = \{5.56, -0.87\}^\top$ is returned as the parameter of the model for t_2.

1) Adaptive Learning Complexity: We can precompute once the nearest neighbors for all tuples in r with cost $O((m+n)n^2) = O(n^3)$ and directly use them in learning individual model for a certain ℓ. According to Algorithm 1, the learning phase computes ϕ_i with cost $O(m^2\ell + m^3)$ for a certain ℓ and cost $O(m^2n^2)$ for all possible ℓ from 1 to n. For each tuple t_i, the cost for computing difference is $O(kn)$. Thus the time cost from Line 3 to Line 7 is $O(kn^2)$. Obviously, it costs $O(n^2)$ to find the proper ℓ^* for all the tuples. Finally, the time complexity of Algorithm 3 is $O(m^2n^2 + n^3)$.

2) Approximation via Stepping: When considering various ℓ in Line 1 in Algorithm 3, instead of increasing 1 in each iteration, i.e., $\ell = \ell + 1$, we may increase more, say $\ell = \ell + h, h \geq 1$ in stepping. The time cost by stepping significantly reduces, from $O(m^2n^3)$ to $O(m^2n^3/h)$. However, it may miss a better model in between ℓ and $\ell + h$. Therefore, stepping is a tradeoff between efficiency and accuracy. (See Section VI-C4 for results under various stepping h.)

Example 5. *For stepping $h = 3$, only the ℓ values $\{1, 4, 7\}$ will be considered, instead of all 8 possible ℓ. Similar to Example 4, for tuple t_2, it computes $cost[2][1] = 3.73, cost[2][4] = 0.09, cost[2][7] = 3.03$. Finally, $\ell_2^* = 4$ is selected and $\Phi_2 = \{5.56, -0.87\}^\top$ is returned.*

B. Incremental Computation

For a specific ℓ, Line 2 in Algorithm 3 calls the individual Learning Algorithm 1 starting from scratch, without utilizing any results from the previous learning, e.g., $\ell - 1$. It is worth noting that the $\ell - 1$ learning neighbors of a tuple are always subsumed in the corresponding ℓ neighbors (Formula 13). Intuitively, the learning computation on $\ell - 1$ neighbors has no need to repeat in the learning over ℓ neighbors.

1) Incremental Learning: Let $T_i^{(\ell)} = \mathsf{NN}(t_i, \mathcal{F}, \ell) = \{t_1, \dots, t_\ell\}$ denote the set of ℓ nearest neighbors of $t_i \in r$, and $\phi_i^{(\ell)}$ be the parameter of the individual regression learned from $T_i^{(\ell)}$ by Formula 5. As aforesaid, subsumption relationship exists among the sets of nearest neighbors with different sizes ℓ. That is, for any tuple $t_i \in r$, $h \geq 1$, we have

$$T_i^{(\ell)} = \mathsf{NN}(t_i, \mathcal{F}, \ell) \subset T_i^{(\ell+h)} = \mathsf{NN}(t_i, \mathcal{F}, \ell + h). \quad (13)$$

Intuitively, the regression model, e.g., $\phi_i^{(\ell+h)}$ learned over $T_i^{(\ell+h)}$, can be incrementally computed from the previous results, i.e., $\phi_i^{(\ell)}$ learned over $T_i^{(\ell)}$, in Proposition 3, rather than starting from scratch in Algorithm 1. Remarkably, we show in Table III that the incremental computation reduces the learning complexity from linear to constant (in terms of the number ℓ).

Let $T_i^{(\ell+h)} = \mathsf{NN}(t_i, \mathcal{F}, \ell + h) = \{t_1, \dots, t_\ell, t_{\ell+1}, \dots, t_{\ell+h}\}$, having

$$T_i^{(\ell+h)} \setminus T_i^{(\ell)} = \{t_{\ell+1}, \dots, t_{\ell+h}\}. \quad (14)$$

To represent the increment, we rewrite $\boldsymbol{Y}^{(\ell+h)}$ in Formula

6 and $\boldsymbol{X}^{(\ell+h)}$ in Formula 7 as follows,

$$\boldsymbol{Y}^{(\ell+h)} = \begin{pmatrix} \boldsymbol{Y}^{(\ell)} \\ t_{\ell+1}[A_m] \\ \vdots \\ t_{\ell+h}[A_m] \end{pmatrix} = \begin{pmatrix} \boldsymbol{Y}^{(\ell)} \\ \boldsymbol{Y}^{(\ell,\Delta_h)} \end{pmatrix}, \tag{15}$$

$$\boldsymbol{X}^{(\ell+h)} = \begin{pmatrix} & \boldsymbol{X}^{(\ell)} & & \\ 1 & t_{\ell+1}[A_1] & \dots & t_{\ell+1}[A_{m-1}] \\ 1 & t_{\ell+2}[A_1] & \dots & t_{\ell+2}[A_{m-1}] \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_{\ell+h}[A_1] & \dots & t_{\ell+h}[A_{m-1}] \end{pmatrix} = \begin{pmatrix} \boldsymbol{X}^{(\ell)} \\ \boldsymbol{X}^{(\ell,\Delta_h)} \end{pmatrix}, \tag{16}$$

where $\boldsymbol{Y}^{(\ell+h)}$ is an $(\ell+h)\times 1$ matrix, and $\boldsymbol{X}^{(\ell+h)}$ is an $(\ell+h)\times m$ matrix.

To incrementally compute $\phi_i^{(\ell+h)}$ by Formula 5, we define

$$\boldsymbol{U}^{(\ell+h)} = (\boldsymbol{X}^{(\ell+h)})^\top \boldsymbol{X}^{(\ell+h)}, \tag{17}$$

$$\boldsymbol{V}^{(\ell+h)} = (\boldsymbol{X}^{(\ell+h)})^\top \boldsymbol{Y}^{(\ell+h)}, \tag{18}$$

where $\boldsymbol{U}^{(\ell+h)}$ is an $m\times m$ matrix, and $\boldsymbol{V}^{(\ell+h)}$ is an $m\times 1$ matrix with both sizes independent of ℓ and h.

Formula 5 for learning the parameter can be rewritten by

$$\phi_i^{(\ell+h)} = (\boldsymbol{U}^{(\ell+h)} + \alpha \boldsymbol{E})^{-1} \boldsymbol{V}^{(\ell+h)}. \tag{19}$$

We show in the proposition below that $\boldsymbol{U}^{(\ell+h)}$ and $\boldsymbol{V}^{(\ell+h)}$ can be incrementally computed from $\boldsymbol{U}^{(\ell)}$ and $\boldsymbol{V}^{(\ell)}$, together with $\boldsymbol{Y}^{(\ell,\Delta_h)}$ and $\boldsymbol{X}^{(\ell,\Delta_h)}$ defined in Formulas 15 and 16.

Proposition 3. *$\boldsymbol{U}^{(\ell+h)}$, $\boldsymbol{V}^{(\ell+h)}$ could be incrementally computed from $\boldsymbol{U}^{(\ell)}$, $\boldsymbol{V}^{(\ell)}$, having*

$$\boldsymbol{U}^{(\ell+h)} = \boldsymbol{U}^{(\ell)} + (\boldsymbol{X}^{(\ell,\Delta_h)})^\top \boldsymbol{X}^{(\ell,\Delta_h)} \tag{20}$$

$$\boldsymbol{V}^{(\ell+h)} = \boldsymbol{V}^{(\ell)} + (\boldsymbol{X}^{(\ell,\Delta_h)})^\top \boldsymbol{Y}^{(\ell,\Delta_h)} \tag{21}$$

where $\ell \in [1, n)$ and $h \in [1, n-\ell]$. (See proof in [1].)

Example 6. *Suppose that learning on t_1 with $\ell = 3$ has been performed, having $\mathsf{NN}(t_1, \{A_1\}, 3) = \{t_1, t_2, t_3\}$,*

$$\boldsymbol{U}^{(3)} = (\boldsymbol{X}^{(3)})^\top \boldsymbol{X}^{(3)} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0.8 & 1.9 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 0.8 \\ 1 & 1.9 \end{pmatrix},$$

$$\boldsymbol{V}^{(3)} = (\boldsymbol{X}^{(3)})^\top \boldsymbol{Y}^{(3)} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0.8 & 1.9 \end{pmatrix} \begin{pmatrix} 5.8 \\ 4.6 \\ 3.8 \end{pmatrix},$$

$$\phi_1^{(3)} = (\boldsymbol{U}^{(3)} + \alpha \boldsymbol{E})^{-1} \boldsymbol{V}^{(3)} = \begin{pmatrix} 5.66 \\ -1.03 \end{pmatrix}.$$

Now we want to learn the parameter $\phi_1^{(4)}$ of t_1 for $\ell = 4$, having $\mathsf{NN}(t_1, \{A_1\}, 4) = \mathsf{NN}(t_1, \{A_1\}, 3) \cup \{t_4\}$. Instead of recomputing entirely the matrices $\boldsymbol{U}^{(4)}$, $\boldsymbol{V}^{(4)}$, they can be incrementally computed from $\boldsymbol{U}^{(3)}$, $\boldsymbol{V}^{(3)}$. Specifically, given $\boldsymbol{X}^{(3,1)} = \begin{pmatrix} 1 & 2.9 \end{pmatrix}$ and $\boldsymbol{Y}^{(3,1)} = \begin{pmatrix} 3.2 \end{pmatrix}$, we have

$$\boldsymbol{U}^{(4)} = \boldsymbol{U}^{(3)} + (\boldsymbol{X}^{(3,1)})^\top \boldsymbol{X}^{(3,1)} = \boldsymbol{U}^{(3)} + \begin{pmatrix} 1 & 2.9 \\ 2.9 & 8.41 \end{pmatrix},$$

$$\boldsymbol{V}^{(4)} = \boldsymbol{V}^{(3)} + (\boldsymbol{X}^{(3,1)})^\top \boldsymbol{Y}^{(3,1)} = \boldsymbol{V}^{(3)} + \begin{pmatrix} 3.2 \\ 9.28 \end{pmatrix},$$

$$\phi_1^{(4)} = (\boldsymbol{U}^{(4)} + \alpha \boldsymbol{E})^{-1} \boldsymbol{V}^{(4)} = \begin{pmatrix} 5.56 \\ -0.87 \end{pmatrix}.$$

TABLE III: Time complexity for learning parameter $\phi_i^{(\ell+h)}$

Computing	From scratch	Incremental
$\boldsymbol{U}$	$m^2(\ell+h)$	$m^2 h$
$\boldsymbol{V}$	$m(\ell+h)$	mh
$(\boldsymbol{U})^{-1}$	m^3	m^3
$(\boldsymbol{U})^{-1}\boldsymbol{V}$	m^2	m^2

TABLE IV: Dataset summary

Dataset	$\lvert r\rvert$	$\lvert\mathcal{R}\rvert$	Source	Property
ASF	1.5k	6	UCI	no clear global regression
CCS	1k	6	UCI	
CCPP	10k	5	UCI	
SN	100k	2	UCI	
PHASE	10k	4	Siemens	a clear global regression
CA	20k	9	KEEL	sparse with high dimension
DA	7k	6	KEEL	
MAM	1k	5	KEEL	real missing, no truth
HEP	200	19	KEEL	real missing, no truth

2) Incremental Learning Algorithm: We revise Algorithm 1 for incremental learning. For each $t_i \in r$, $T_i^{(\ell+h)} \setminus T_i^{(\ell)}$ is retrieved in Formula 14, rather than all the ℓ nearest neighbors in Line 2 in Algorithm 1. Referring to Proposition 3, we incrementally compute $\boldsymbol{U}^{(\ell+h)}$, $\boldsymbol{V}^{(\ell+h)}$ from $\boldsymbol{U}^{(\ell)}$, $\boldsymbol{V}^{(\ell)}$, together with $\boldsymbol{Y}^{(\ell,\Delta_h)}$ and $\boldsymbol{X}^{(\ell,\Delta_h)}$ on nearest neighbor increments. Finally, $\phi_i^{(\ell+h)}$ is computed by Formula 19.

It is worth noting that incrementally computing $\phi_i^{(\ell+h)}$ only needs to cache $\boldsymbol{U}^{(\ell)}$, $\boldsymbol{V}^{(\ell)}$ in the previous step. Earlier results such as $\boldsymbol{U}^{(\ell-h)}$, $\boldsymbol{V}^{(\ell-h)}$ could be discarded. Given the same h, the incremental computation naturally supports stepping.

3) Complexity Analysis: Table III lists the major steps and costs for learning parameter $\phi_i^{(\ell+h)}$ in Formula 19. As shown, the costs of computing $\boldsymbol{U}$ and $\boldsymbol{V}$ from scratch using Formulas 17 and 18 are linear in terms of ℓ. With the incremental computation in Formulas 20 and 21 in Proposition 3, the costs become irrelevant to ℓ. In other words, we reduce the learning cost from linear $O(m^2\ell + m^2 h + m^3)$ to constant $O(m^2 h + m^3)$ in terms of ℓ tuples.

VI. Experiment

While the theoretical analysis in Section IV proves that our proposal subsumes some existing methods, the empirical evaluation particularly concerns how IIM outperforms the existing imputation approaches in practice, in Section VI-B.

A. Settings

1) Datasets: We employ 9 datasets from different sources, UCI[1] [21], KEEL[2] [2] and Siemens, with various properties as summarized in Table IV. For instance, no clear linear regression is observed globally in the ASF dataset, i.e., with heterogeneity problem, while the PHASE dataset has a clear regression relationship in three-phase electric power. The CA dataset involves 9 attributes with higher dimension, which leads to more serious sparsity issue. The MAM and HEP datasets contain real-world missing values without ground truth, and are used for evaluating the classification application with / without imputation (in [1]).

2) Criteria: Following the same line of evaluating data quality approaches [5], for each dataset (except the two datasets without ground truth), we randomly select a set of tuples as $\{t_x\}$ by removing values on (multiple) attributes $\{A_x\}$ as missing values. The remaining tuples are considered as complete tuples in r. When multiple incomplete attributes $\{A_x\}$ exist, we impute them one by one. RMS error [18] is employed to evaluate the imputation accuracy, $\sqrt{\frac{\sum_{t_x,A_x}(t_x[A_x]-t'_x[A_x])^2}{|\{(t_x,A_x)\}|}}$, where $t_x[A_x]$ is the original value (ground truth) of the incomplete attribute, and $t'_x[A_x]$ is the corresponding imputation. The lower the RMS error is, the better the imputation accuracy will be, i.e., closer to the truth.

The sparsity issue states that a tuple does not have sufficient neighbors that share the same/similar values. In other words, the truth value varies from the values suggested by complete neighbors. To evaluate the variance, we employ the *coefficient of determination* [12], $R^2 = 1 - \frac{\sum_{t_x}(t_x[A_m]-t'_x[A_m])^2}{\sum_{t_x}(t_x[A_m]-\overline{t_i[A_m]})^2}$, where $t_i \in r$, $t_x[A_m]$ is the truth value, and $t'_x[A_m]$ is the value suggested by complete neighbors (e.g., by kNN). We denote R^2_S the R^2 measure on sparsity. The lower the measure R^2_S is, the more serious the sparsity issue will be in the data.

The heterogeneity issue states that tuples do not fit a single global model. Similarly, we evaluate how the truth value varies from the values predicted by the single global model. Again, the aforesaid *coefficient of determination* is employed, where $t'_x[A_m]$ is the value predicted by the single global model (e.g., by GLR). The lower the measure R^2_H is, the more serious the heterogeneity issue will be in the data.

B. Comparison on Imputation Methods

This experiment compares our proposal IIM with the existing approaches listed in Table II in Section II. We use the MICE implementation[3] of PMM and BLR in R, the XGB implementation in R, and the existing SVD implementation[4]. Other approaches as well as our IIM are implemented in Java. Thereby, the corresponding time costs could be compared, e.g., in Figures 7 and 9. While some significantly worse results may not appear in the figures, the results of all methods can be found in Tables V and VI.

[1]http://archive.ics.uci.edu/ml/datasets/

[2]http://sci2s.ugr.es/keel/datasets.php

[3]https://github.com/stefvanbuuren/mice/tree/master/R

[4]https://github.com/jeffwong/imputation

1) Imputation on Various Datasets: For each dataset in Table V, we randomly pick 5% tuples as t_x with one missing value on a random attribute A_x. That is, there are $5\%\frac{1}{|\mathcal{R}|}$ missing values w.r.t. the total values in each dataset, where $|\mathcal{R}|$ is the number of attributes in the dataset. For instance, the CCPP data with 5 attributes has $\frac{5\%}{5} = 1\%$ missing values.

When a dataset is with high sparsity but low heterogeneity, i.e., small R^2_S but large R^2_H, such as CA in Table V, the GLR approach using the predicted value via the regression model shows a better imputation performance (RMS=0.6) than the kNN method using the (aggregated) value in the complete neighbor tuples (RMS=2.02).

Nevertheless, our proposed IIM always shows the lowest imputation error. The result is not surprising referring to the theoretical analysis in Section IV that our proposal subsumes GLR and kNN as special cases.

To show applicability, we report the results on the larger dataset SN in Table V. As shown, the better imputation result of our proposed IIM is still consistently observed. (The results of SVD, ILLS and XGB are not available since they cannot be implemented on only two attributes.)

2) Varying the Missing Attribute A_x: Table VI reports the results on various incomplete attributes A_x over the ASF data. Owing to the different ranges of domain values on various attributes, the imputation RMS error differs in attributes.

Approaches perform variously over the attributes with different domain characteristics in terms of sparsity and heterogeneity. In Table VI, for attribute A_4 with small R^2_S (high sparsity) but large R^2_H (low heterogeneity), the attribute model methods (GLR and LOESS using the values predicted by regression models) perform better than the tuple model methods (kNN using the aggregated value of complete neighbor tuples). In contrast, for attribute A_6 with large R^2_S (low sparsity) but small R^2_H (high heterogeneity), kNN outperforms GLR. Nevertheless, since our proposal concerns both sparsity and heterogeneity, IIM consistently shows the best performance. The results verify the superiority of our proposal.

3) Varying the Number of Complete Attributes $|\mathcal{F}|$: When preparing the datasets, we randomly pick a certain percent (%) tuples as t_x with one missing value on a random attribute A_x. By default, all the remaining attributes are used as complete neighbors for imputation, i.e., $\mathcal{F} = \mathcal{R} \setminus \{A_x\}$. In order to evaluate the imputation with different sizes of complete attributes, the experiments in Figures 4 and 5 consider a subset of $\mathcal{R} \setminus \{A_x\}$ as the complete attributes $\mathcal{F}$. For instance, a number of complete attributes $|\mathcal{F}| = 2$ in the x axis denotes $\mathcal{F} = \{A_1, A_2\}$, instead of considering all the attributes in $\mathcal{R} \setminus \{A_x\} = \{A_1, A_2, A_3, \dots\}$ as complete attributes.

Figures 4 and 5 present the results on various number of complete attributes $|\mathcal{F}|$. For most approaches, it is not surprising that imputation improves under more complete attributes. Specifically, with more attributes in $\mathcal{F}$, the regression from $\mathcal{F}$ to A_x will be more reliable (if exists). Furthermore, the neighbors found w.r.t. larger $\mathcal{F}$ are more likely to share values. With both aforesaid benefits, our IIM shows more significant improvements when $\mathcal{F}$ is large.

TABLE V: Imputation RMS error of IIM compared to the existing approaches listed in Table II over various datasets

Dataset	R_S^2	R_H^2	IIM	kNN	kNNE	IFC	GMM	SVD	ILLS	GLR	LOESS	BLR	ERACER	PMM	XGB
ASF	0.85	0.73	**8.08**	22.63	20.12	50.72	59.04	37.88	16.05	30.28	16.73	42.78	20.35	36.43	11.61
CA	0.03	0.90	**0.49**	2.02	1.85	2.03	2.12	50.11	12.76	0.6	0.54	0.88	0.6	0.77	0.7
CCPP	0.95	0.93	**3.75**	3.98	4.13	14.08	23.09	6.79	5.78	4.58	4.25	6.55	3.97	6.19	4.45
CCS	0.63	0.56	**10.45**	12.84	11.13	21.39	24.95	25.59	13.67	13.64	12.76	20.51	11.25	18.85	11.26
DA	0.65	0.68	**15.52**	16.99	17.75	22.92	23.99	21.92	94.5	16.68	15.88	23.69	16.18	23.47	**15.56**
PHASE	0.9	0.91	**3.31**	3.51	3.42	5.41	11.35	5.28	3.59	**3.32**	**3.32**	4.73	**3.32**	4.64	**3.36**
SN	0.79	0.05	**0.11**	**0.12**	**0.12**	0.28	0.43	-	-	0.27	0.20	0.4	**0.13**	0.28	-

TABLE VI: Imputation RMS error on various incomplete attribute A_x over ASF dataset with 100 incomplete tuples

	R_S^2	R_H^2	IIM	kNN	kNNE	IFC	GMM	SVD	ILLS	GLR	LOESS	BLR	ERACER	PMM	XGB
A_1	0.47	0.46	**192.5**	235.2	247.8	326.9	334.8	320.4	248.3	234.4	201.8	328.5	206.8	289.9	204.9
A_2	0.85	0.73	**8.08**	22.63	20.12	50.72	59.04	37.88	16.05	30.28	16.73	42.78	20.35	36.43	11.61
A_3	0.73	0.5	**1.49**	5.11	4.08	8.87	12.15	9.67	4.73	6.54	3.66	9.18	4.51	8.72	2.07
A_4	0.03	0.12	**12.82**	15.74	13.28	15.65	16.74	15.16	17.62	14.68	13.84	21.14	14.68	20.23	13.24
A_5	0.79	0.63	**13.85**	64.94	60.29	125.58	138.22	88.76	34.9	80.54	55.95	116.65	58.01	90.53	23.23
A_6	0.78	0.51	**3.22**	3.28	4.59	6.39	7.39	45.25	11.82	4.8	3.4	7.02	**2.92**	6.29	15.25

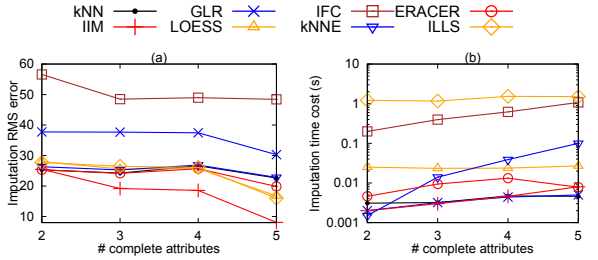


Fig. 4: Varying the number of complete attributes $|\mathcal{F}|$, over ASF with 100 incomplete tuples

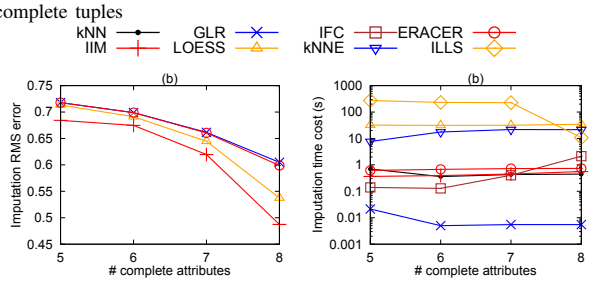


Fig. 5: Varying the number of complete attributes $|\mathcal{F}|$, over CA with 1k incomplete tuples

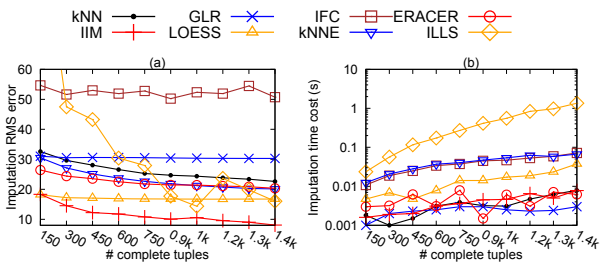


Fig. 6: Varying the number of complete tuples $n = |r|$, over ASF with 100 incomplete tuples

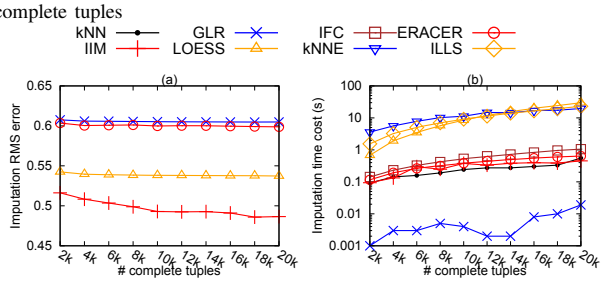


Fig. 7: Varying the number of complete tuples $n = |r|$, over CA with 1k incomplete tuples

Figures 4(b) and 5(b) report the time cost of IIM in the imputation phase (the offline learning phase only needs to be processed once for imputing different incomplete tuples). In contrast, LOESS and ILLS need to online learn the local regression over the neighbors of the input incomplete tuple, and thus have high imputation time cost. It is not surprising that IIM shows similar time cost as kNN, since both approaches need to find k nearest neighbors.

4) Varying the Number of Complete Tuples $n = |r|$: Figures 6 and 7 report the results by randomly selecting n tuples from the dataset as r of complete tuples. Generally, more complete tuples lead to better imputation performance. The interesting result in Figure 6(a) is that kNN relies more on complete tuples to achieve lower imputation error, since it requires the presence of sufficient neighbors sharing similar values. Our IIM utilizing the individual regressions of tuples benefits from more complete tuples as well.

5) Varying the Cluster Size of Incomplete Tuples: Rather than introducing missing values in random tuples, we consider incomplete tuples that cluster together. That is, complete neighbors are very far away. Figure 8 reports the results under various sizes of incomplete tuple clusters. For example,

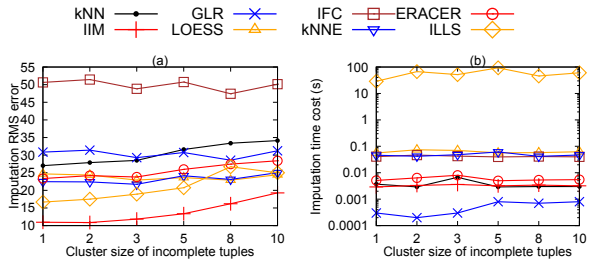


Fig. 8: Varying the cluster size of incomplete tuples, over ASF with 100 incomplete tuples in total

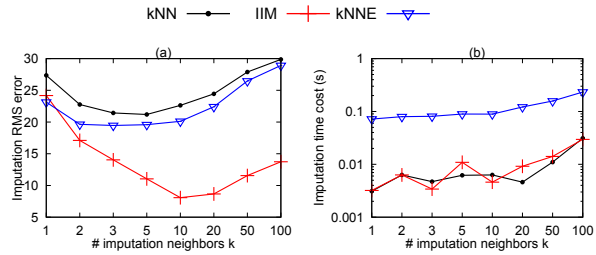


Fig. 9: Varying the number of imputation neighbors k, over ASF with 100 incomplete tuples

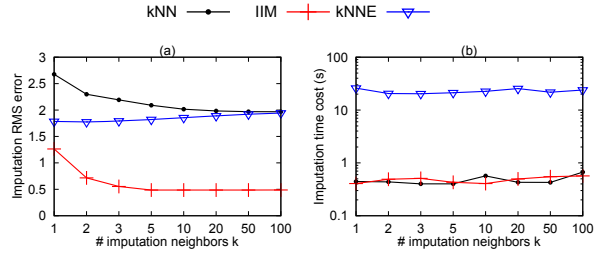


Fig. 10: Varying the number of imputation neighbors k, over CA with 1k incomplete tuples

a cluster size 3 denotes that the 2 closest neighbors are also incomplete tuples. It is not surprising that with the increase of incomplete tuple cluster size, all the tuple model based imputation methods relying on the closest neighbors (e.g., kNN, ILLS) become worse. On the other hand, the attribute model based methods (such as GLR or LOESS) are relatively stable. Again, our proposed IIM still shows the best performance, since it does not rely on the neighbor tuples to share the same values, and thus can cope with the sparsity issue introduced by the clusters of incomplete tuples.

C. Evaluation on Individual Learning

In this section, we evaluate the characteristic of proposed techniques on the following aspects to show the performance and rational behind IIM.

1) Varying the Number of Imputation Neighbors k: This experiment evaluates various number of imputation neighbors k. It is used in both kNN, kNNE and our IIM (in Algorithm 2 of imputation phase). Figures 9 and 10 report the results on ASF (having heterogeneity issues) and CA (having sparsity property) with 5% incomplete tuples. Generally, a moderately large k is preferred. If k is too small, it is not reliable to

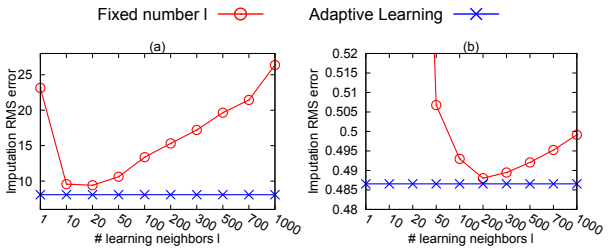


Fig. 11: Comparison between adaptive learning and the learning over various fixed number ℓ of learning neighbors, over (a) ASF and (b) CA

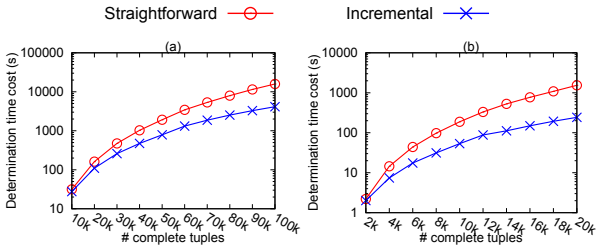


Fig. 12: Scalability of adaptive learning (with straightforward and incremental computation) on the number n of tuples in r, over (a) SN and (b) CA

support the imputation. On the other hand, if k is too large, irrelevant tuples may distract the imputation, as illustrated in Figure 9(a). For the CA data with sparsity issue in Figure 10(a), changing the number of neighbors k does not help much in imputation. (Some significantly worse results do not appear in the figure, such as kNN as shown in Table V.)

2) Evaluating Adaptive Learning: This experiment evaluates two aspects: (1) how the fixed number ℓ of learning neighbors for all tuples in Algorithm 1 affects the imputation results; and (2) does the adaptive learning with distinct number of learning neighbors for different tuple in Algorithm 3 truly improve the imputation?

First, as shown in Figure 11, a small number ℓ of learning neighbors may suffer from the overfitting problem and lead to poor imputation. On the other hand, when ℓ is too large, the learned individual model may suffer from the heterogeneity problem (under-fitting) and hence also has bad performance. Manually choosing a proper ℓ is non-trivial, which is very different from datasets as illustrated in Figures 11 (a) and (b).

Nevertheless, the proposed Adaptive Learning Algorithm 3 can successfully address this problem, by adaptively considering a distinct number ℓ of learning neighbors for each tuple individually. As illustrated in Figure 11, the performance of adaptive learning is better than setting a fixed ℓ for all tuples.

3) Evaluating Incremental Learning: Figure 12 reports the time cost of adaptive learning using straightforward and incremental computation (with stepping $h = 50$) under various number n of tuples in r. The incremental learning algorithm devised in Section V-B2 shows up to one order of magnitude improvement compared to the straightforward adaptive learning Algorithm 3. The result is not surprising, since the incremental computation reduces the time cost of parameter learning from linear to constant (in terms of ℓ), as shown in

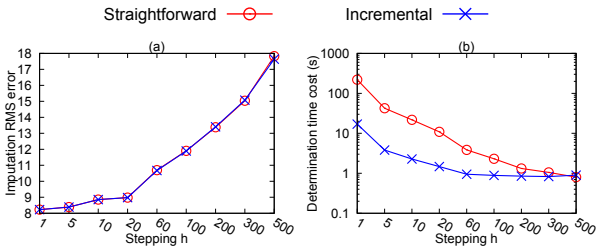


Fig. 13: Varying stepping h over ASF

Table III in Section V-B3. To show scalability, we report time cost of adaptive learning on SN in Figure 12(a). Again, the result is generally similar to the CA dataset with 20k tuples.

4) Tradeoff via Stepping: Figure 13 present the results on varying the stepping h studied in Section V-A2. The smaller the h is, the more the candidate ℓ values are considered. When $h = 1$, all the possible ℓ values are evaluated. It is not surprising that a small stepping h with more candidate ℓ values considered leads to lower imputation error in Figure 13(a), while the corresponding time cost is higher in Figure 13(b). The exactly same imputation errors of straightforward and incremental determination algorithms verify the correctness of incremental computation. Figure 13(b) demonstrates again the significant improvement in time cost by the incremental determination algorithm.

VII. Conclusions

To cope with the challenges of sparsity (no sufficient similar neighbors) and heterogeneity (tuples do not fit the same regression model) in imputing numerical data, we propose IIM, Imputation via Individual Models. The rationale of our proposal is illustrated first by theoretically proving that some existing approaches are indeed special cases of IIM under extreme settings (i.e., $\ell = 1$ or $\ell = n$ in Propositions 1 and 2). It further motivates us to select a proper number ℓ of learning neighbors (in between the extreme 1 and n) to avoid over-fitting or under-fitting. Again, owing to the heterogeneity issue, the number ℓ of learning neighbors could be different for learning the individual models of different tuples. Through a validation step, we adaptively determine a model for each complete tuple that can best impute other tuples (in validation). Efficient incremental computation is devised for adaptive learning, where the time complexity of learning a model reduces from linear to constant. Experiments on read data demonstrate the superiority of our proposal.

Acknowledgement: This work is supported by National Key Research Program of China under Grant 2016YFB1001101; China NSFC under Grants 61572272 and 71690231.

References

[1] Full version. *http://ise.thss.tsinghua.edu.cn/sxsong/doc/missing.pdf.*
[2] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, and S. García. KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287, 2011.
[3] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.
[4] C. Anagnostopoulos and P. Triantafillou. Scaling out big data missing value imputations: pythia vs. godzilla. In *SIGKDD*, pages 651–660, 2014.
[5] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, and D. Santoro. Messing up with BART: error generation for evaluating data-cleaning algorithms. *PVLDB*, 9(2):36–47, 2015.
[6] G. E. Batista, M. C. Monard, et al. A study of k-nearest neighbour as an imputation method. *HIS*, 87(251-260):48, 2002.
[7] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755, 2007.
[8] S. v. Buuren and K. Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of statistical software*, pages 1–68, 2010.
[9] Z. Cai, M. Heydari, and G. Lin. Microarray missing value imputation by iterated local least squares. In *Proceedings of 4th Asia-Pacific Bioinformatics Conference.*, pages 159–168, 2006.
[10] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *SIGKDD*, pages 785–794, 2016.
[11] W. S. Cleveland and C. Loader. Smoothing by local regression: Principles and methods. In *Statistical theory and computational aspects of smoothing*, pages 10–49. Springer, 1996.
[12] J. L. Devore. *Probability and Statistics for Engineering and the Sciences*. Cengage learning, 2011.
[13] A. Doan, A. Y. Halevy, and Z. G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.
[14] C. Domeniconi and B. Yan. Nearest neighbor ensemble. In *17th International Conference on Pattern Recognition, ICPR 2004, Cambridge, UK, August 23-26, 2004.*, pages 228–231, 2004.
[15] A. Farhangfar, L. A. Kurgan, and W. Pedrycz. A novel framework for imputation of missing values in databases. *IEEE Trans. Systems, Man, and Cybernetics, Part A*, 37(5):692–709, 2007.
[16] D. Firmani, M. Mecella, M. Scannapieco, and C. Batini. On the meaningfulness of "big data quality" (invited paper). *Data Science and Engineering*, 1(1):6–20, 2016.
[17] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. A pipelined framework for online cleaning of sensor data streams. In *ICDE*, page 140, 2006.
[18] S. R. Jeffery, M. N. Garofalakis, and M. J. Franklin. Adaptive cleaning for RFID data streams. In *VLDB*, pages 163–174, 2006.
[19] L. R. Landerman, K. C. Land, and C. F. Pieper. An empirical evaluation of the predictive mean matching method for imputing missing values. *Sociological Methods & Research*, 26(1):3–33, 1997.
[20] D. Li, J. S. Deogun, W. Spaulding, and B. Shuart. Towards missing data imputation: A study of fuzzy k-means clustering method. In *RSCTC*, volume 3066 of *Lecture Notes in Computer Science*, pages 573–579, 2004.
[21] M. Lichman et al. Uci machine learning repository, 2013.
[22] R. J. Little. Regression with missing x's: a review. *Journal of the American Statistical Association*, 87(420):1227–1237, 1992.
[23] R. J. Little and D. B. Rubin. *Statistical analysis with missing data*, volume 333. John Wiley & Sons, 2014.
[24] C. Mayfield, J. Neville, and S. Prabhakar. ERACER: a database approach for statistical inference and data cleaning. In *SIGMOD*, pages 75–86, 2010.
[25] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. Adaptive computation and machine learning. MIT Press, 2012.
[26] S. Nikfalazar, C. Yeh, S. E. Bedingfield, and H. A. Khorshidi. A new iterative fuzzy clustering algorithm for multiple imputation of missing data. In *FUZZ-IEEE*, pages 1–6. IEEE, 2017.
[27] C. R. Rao, C. R. Rao, M. Statistiker, C. R. Rao, and C. R. Rao. *Linear statistical inference and its applications*, volume 2. Wiley New York, 1973.
[28] D. B. Rubin. *Multiple imputation for nonresponse in surveys*, volume 81. John Wiley & Sons, 2004.
[29] B. Saha and D. Srivastava. Data quality: The other face of big data. In *ICDE*, pages 1294–1297, 2014.
[30] O. G. Troyanskaya, M. N. Cantor, G. Sherlock, P. O. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman. Missing value estimation methods for DNA microarrays. *Bioinformatics*, 17(6):520–525, 2001.
[31] X. Yan, W. Xiong, L. Hu, F. Wang, and K. Zhao. Missing value imputation based on gaussian mixture model for the internet of things. *Mathematical Problems in Engineering*, 2015, 2015.