2021 IEEE 37th International Conference on Data Engineering (ICDE)

Capturing Semantics for Imputation with Pre-trained Language Models

Yinan Mei [†], Shaoxu Song [†], Chenguang Fang [†], Haifeng Yang [§], Jingyun Fang [§], Jiang Long [§]
[†] *BNRist, School of Software, Tsinghua University*, [§] *Data Governance Innovation Lab, HUAWEI Cloud BU*
{myn18@mails., sxsong@, fcg19@mails.}tsinghua.edu.cn, {yanghaifeng8, fangjingyun, longjiang4}@huawei.com

***Abstract*—Existing imputation methods generally generate several possible fillings as candidates and determine the value from the candidates for imputing. However, semantics are ignored in these methods. Recently, pre-trained language models achieve good performances in various language understanding tasks. Motivated by this, we propose IPM that captures semantics for Imputation with Pre-trained language Models. A straightforward idea is to model the imputation task as a multiclass classfication task, named IPM-Multi. IPM-Multi predicts the missing values by fine-tuning the pre-trained model. Due to the low redundancy of databases and large domain sizes, IPM-Multi may suffer the over-fitting problem. In this case, we develop another approach named IPM-Binary. IPM-Binary first generates a set of uncertain candidates and fine-tunes a pre-trained language model to select candidates. Specifically, IPM-Binary models the candidate selection task as a binary classification problem. Unlike IPM-Multi, IPM-Binary computes the probability for each candidate filling respectively, by accepting both complete attributes and a candidate filling as input. The attention mechanism enhances the ability of IPM-Binary to capture semantic information. Moreover, negative sampling from neighbors rather than domains is employed to accelerate the training process and makes the training more targeted and effective. As a result, IPM-Binary requires fewer data to converge. We compare our proposal IPM to the state-of-the-art baselines on multiple datasets. And the extensive experimental results show that IPM outperforms existing solutions. The evaluation of IPM validates our intuitions and demonstrates the effectiveness of the proposed optimizations.**

***Index Terms*—Imputation, Deep Learning, Pre-trained Language Models**

I. Introduction

In practice, missing data are prevalent, due to the optional inputs in the information collection system and mismatching in integrating heterogenous data sources, and so on. Obviously, these missing values significantly reduce the quality of the data and make the data hard to use.

A. Challenges

Many existing imputation techniques [2], [31] depend on the symbolic similarity metrics, such as string edit distance and term jaccard similarity. However, relying solely on symbolic similarity does not always lead to accurate filling (see in Example 1). Some imputation techniques [27], [38] inferring corrections (including imputation) from co-occurring attribute values suffer the sparsity problem. The co-occurrence w.r.t values are not always available, as shown in Example 1. Thus, we propose to impute the missing values by further considering semantics w.r.t the fine-grained tokens.

TABLE I
Example relational data with missing values.

r	title	category	modelno	price
t_1	tribeca varsity jacket hard shell case for ipod touch dallas cowboys	Nan	fva3778	18.54
t_2	oklahoma sooners iphone 4 case black shell	electronics general	fva3161	29.99
t_3	belkin sport armband for ipod nano black	mp3 accessories	f8z514tt064	18.88
t_4	case logic 11.6 hard shell netbook sleeve	electronics general	154722	19.99
t_5	fellowes hd precision cordless mouse	mice	98904	44.84
...	...	...	...	...

Example 1. *Consider a part of the real product dataset Walmart*[3] *in Table I. Tuples t_1-t_5 represent five different products with four attributes. There is a missing entry $t_1[A_{category}]$ (we manually label the truth "mp3 accessories").*

The nearest neighbor based imputation, such as kNN [2], finds top-k tuples that are most similar to t_1 on the complete attribute. The major class of the k neighbors on the to-impute attribute "category" will be taken as the imputation for $t_1[A_{category}]$. For example, given $k = 3$ and term jaccard similarity as the similarity metric, kNN will identify t_2, t_3, t_4 as neighbors. And "electronics general", the major class on the attribute "category" of the 3-NNs, will be imputed in $t_1[A_{category}]$. It is wrong given the truth "mp3 accessories".

The statistics-based method HoloClean [27], [38] first generates the candidates from co-occurring values and infers the filling from candidates with statistical models. However, there is no tuple sharing the same attribute value with t_1 and thus the statistical model does not apply. HoloClean fails to find candidates for imputing $t_1[A_{category}]$.

Recently, many approaches in Natural Language Processing (NLP) area are good at capturing semantics. For instance, BERT [7] and GPT [24] achieve excellent performances in many language understanding tasks like Question Answering. BERT [7] is trained to predict the masked token in text, and E-BERT [41] further predicts the masked phrases during pre-training. Nevertheless, these NLP-based approaches have not yet been used for relational data imputation.

A natural idea is that we can take the tuple as a sentence and predict the original value for the missing value via multiclass classfication like Sentiment Analysis task. Owing to the low redundancy of databases and large domain sizes, the model is likely to fall into the pitfall of over-fitting.

B. Our Proposal

We propose an approach named **I**mputation with **P**re-trained language **M**odels (IPM) to address the above issues. Since the pre-trained language model (LM) has been pre-trained on a large corpus such as Wikipedia [7] and Text8 [13], it can capture the semantics of tokens co-occurring in the corpus, as shown in Example 2 below. IPM benefits from the semantics in the pre-trained LMs, for determining the candidates that are more semantically related to the complete values.

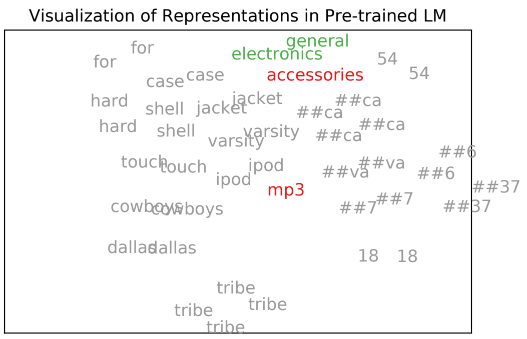

Fig. 1. Visualization of tokens (dimension reduction via t-SNE)

Example 2. *Consider again the missing $t_1[A_{category}]$ in Table I in Example 1. Referring to kNN, we have two candidates "mp3 accessories" and "electronics general", of which "mp3 accessories" is the truth. As shown in Figure 1, the pre-trained LM can capture the correlation between "ipod" in $t_1[A_{title}]$ and "mp3" in the candidate to help find the correct filling.*

As stated in Section III, Our proposal IPM consists of two modules named IPM-Multi and IPM-Binary. IPM-Multi models the imputation problem as the multiclass classfication problem, where the number of classes is the domain size. IPM-Multi can achieve good performance if there are sufficient training data (see Section VI-B1 and Section VI-C1). Due to the low redundancy of relational databases [5] and large domain size, the training data are not always sufficient. In this case, IPM-Multi will suffer the over-fitting problem. In addition, it is suboptimal to take all values in the domain as class labels without semantics. We can take advantage of this semantic relationship to impute the missing value, as shown in Example 1. Since IPM-Multi only considers the candidate filling in the domain as the semantically free class label, this kind of semantic relationship is ignored.

To better capture the semantics and avoid the over-fitting problem, we propose IPM-Binary, which is the focus of our study. Rather than directly predicting the missing value with a multiclass classifier, IPM-Binary determines whether each candidate filling is correct respectively with a binary classifier.

IPM-Binary first finds the neighbors $NN(t_i)$ of the incomplete tuple t_i. For the missing cell $t_i[A_k]$, we collect the values on attribute A_k from its neighbors $NN(t_i)$ as the candidates $can(t_i[A_k])$. Then we fine-tune a pre-trained language model to select the final filling $\phi(t_i[A_k])$ from the candidates.

Compared to IPM-Multi, IPM-Binary requires less training data for two reasons: (1) IPM-Multi needs to learn the features for each specific class. IPM-Binary takes the candidate filling and the complete attribute values as the input and determines whether they are related. The search space of IPM-Binary $can(t_i[A_k])$ is relatively smaller than IPM-Multi $dom(A_k)$. (2) IPM-Binary can capture the semantic relationship between the correct fillings and the complete attributes values with Self-attention mechanism. With the semantic relationship, IPM-Binary can quickly converge with limited training data and determine the filling accurately, as shown in Section VI-C1.

Moreover, we propose to conduct negative sampling from the neighbors rather than from the domain. Recall that we select imputation from candidates generated by neighbors. Thus, the neighbor sampling strategy makes the training more targeted and effective, as discussed in Section V-B3.

C. Contributions

Our major contributions in this study are as follows:

1. We propose a new data imputation approach, named IPM. Compared to the existing imputation methods, IPM can captures semantics in data for imputation with the pre-trained language model (Section III).
2. We propose to model the imputation problem as a multiclass classification problem when training data are sufficient. The number of classes is the domain size (Section IV).
3. We propose selecting the filling from candidates via a novel binary classifier (IPM-Binary), which overcomes the over-fitting problem due to the low redundancy of relational databases (Section V).
4. We propose to conduct negative sampling from neighbors rather than the domain. This sampling strategy makes the fine-tuning more targeted so that the fine-tuned model can better select the correct filling (Section V-B2).
5. We conduct extensive experiments on several datasets in Section VI. The comparison with baselines shows the superiority of our proposal.

II. Preliminary

A. Pre-Trained Language Models

Recently, substantial works [37] show pre-trained language models (pre-trained LMs) are beneficial for downstream tasks in NLP. Pre-trained LMs are trained on a large corpus of text and learn universal language representations. The representations obtained by pre-trained LMs capture the semantic connection between words in the training corpus. Thus, if we perform downstream tasks by fine-tuning pre-trained LMs, the training cost would be lower than training from scratch. In addition, the models would benefit from the information contained in the training corpus of pre-trained LMs.

Pre-trained LMs can be divided into two classes [23]. The first generation of pre-trained LMs, such as Skip-Gram [20] and GloVe [22], aims to learn good representations of words

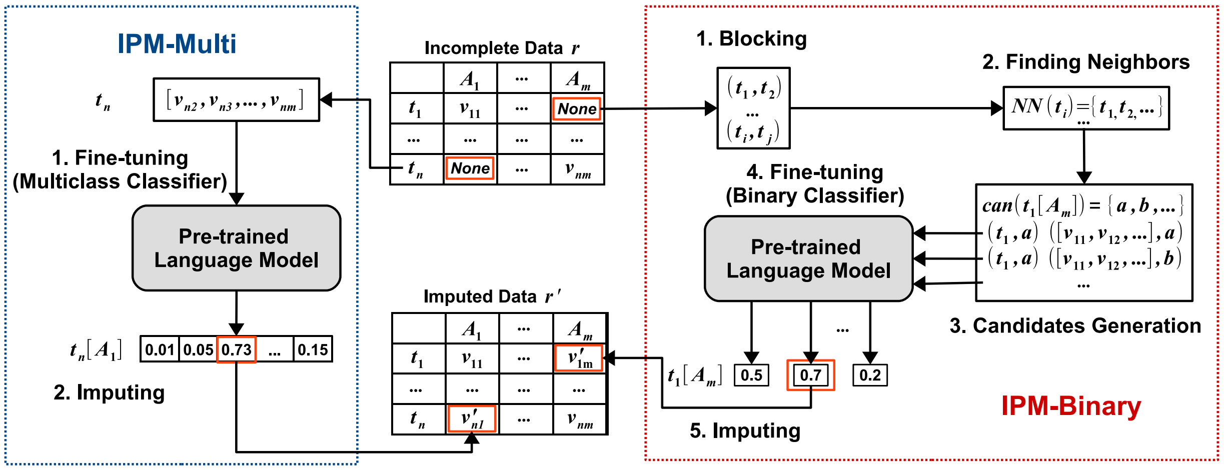
IPM-Multi
t_n
$[v_{n2}, v_{n3}, ..., v_{nm}]$
1. Fine-tuning (Multiclass Classifier)
Pre-trained Language Model
$t_n[A_1]$ 0.01 0.05 0.73 ... 0.15
2. Imputing
Incomplete Data r
Imputed Data r'
1. Blocking
2. Finding Neighbors
$NN(t_i) = \{t_1, t_2, ...\}$
3. Candidates Generation
$can(t_1[A_m]) = \{a, b, ...\}$
4. Fine-tuning (Binary Classifier)
Pre-trained Language Model
$t_1[A_m]$ 0.5 0.7 0.2
5. Imputing
IPM-Binary

Fig. 2. The workflow of IPM. IPM consists of two alternatives, named IPM-Multi and IPM-Binary. (1) IPM-Multi plugs in a multiclass classifier into the pre-trained language model, where the number of classes is the domain size. IPM-Multi fine-tunes the pre-trained LM and the classifier to predict the missing values. (2) IPM-Binary first generates candidate fillings from neighbors. Each candidate filling will be combined with the values of complete attributes as the input. IPM-Binary sets a binary classifier after the pre-trained LM. Then IPM-Binary fine-tunes all the parameters to predict whether the combination is correct, i.e., the candidate filling is true. The filling with the highest probability outputted by the classifier will be taken as the imputation result.

from the co-occurrence/semantic relationship between them. However, these methods fail to capture contextualized relationships, useful in sentence matching [15]. The second generation of pre-trained LMs, such as BERT [7] and RoBERTa [16], focuses on capturing context information. These models are stacked with multiple Transformer blocks [36], which apply fully-connected self-attention layers, typically 12 or 24 layers. The self-attention mechanism [36] is applied in each layer with the self-attention function. The self-attention function can be described as mapping a query and a set of key-value pairs to an output, where query, keys and values are all representations of tokens,

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

where d_k is the dimension of the representation vectors. The self-attention layer takes a sequence of word embeddings as inputs and calculates the correlation between each pair of words. Then the word embeddings would be reconstructed via a weighted sum of the inputs. In this way, the pre-trained LMs successfully learn contextual representations.

With the self-attention mechanism, the pre-trained LMs can be directly fine-tuned for many downstream tasks such as Question Answering and Sequence Classification [7]. Compared to training a task-specific model from scratch, fine-tuning the pre-trained LMs can converge faster and does not require much training data.

B. Problem Settings

Consider a relation instance $r = \{t_1, t_2, ..., t_n\}$ over schema $R = (A_1, A_2, ..., A_m)$. The domain of attribute A_k is $dom(A_k)$. We denote the value of attribute A_k in tuple t_i as $t_i[A_k]$. If $t_i[A_k] = None$, then we say this cell is incomplete. For tuple t_i, we denote complete attributes as $A_C^i = \{A_p, A_q, ...\}$, where $t_i[A_p] \neq None$, $\forall A_p \in A_C^i$. For each incomplete cell $t_i[A_k] = None$, we consider a set of candidates $can(t_i[A_k]) \subseteq dom(A_k)$ for imputing cell $t_i[A_k]$.

If we have a function $p(v, t_i, A_k)$, which can evaluate the probability that the candidate $v \in can(t_i[A_k])$ is the correct imputation. Thus, our problem is to find the candidate $v \in can(t_i[A_k])$, which maximizes $p(v, t_i, A_k)$, i.e.,

$$\phi(t_i[A_k]) = \underset{v \in can(t_i[A_k])}{argmax} \quad p(v, t_i, A_k)$$

where $\phi(t_i[A_k])$ is the imputation for the incomplete $t_i[A_k]$.

III. Overview

Figure 2 illustrates the workflow of IPM. Given an incomplete relation instance, IPM utilizes the pre-trained language model to select the filling for each missing cell. IPM consists of two alternatives named IPM-Multi and IPM-Binary. When we have sufficient training data, IPM-Multi can achieve good performance. Otherwise, IPM-Binary is a better choice.

A. IPM-Multi

IPM-Multi models the imputation problem as a multiclass classfication task. The missing values will be predicted via a softmax network, where the candidates are the domain $dom(A_k)$. Due to the large domain size, IPM-Multi requires abundant data for training. IPM conducts 2 steps in IPM-Multi.

Step 1: Fine-tuning. IPM-Multi fine-tunes the pre-trained language model to predict the original value of the missing cell, as shown in Section IV-A.

Step 2: Imputing. After fine-tuning, IPM-Multi will select the value corresponding to the class with the maximum probability output by softmax as the imputation result.

B. *IPM-Binary*

Without sufficient training data, IPM-Multi is likely to fall into the over-fitting pitfall. Rather than directly selecting the imputation value from the whole domain, IPM-Binary evaluates the probability for a candidate value each time via a binary classifier as stated in Section V-B. IPM-Binary requires less training data compared to IPM-Multi. The IPM-Binary module consists of 5 steps as follows:

Step 1, 2 and 3: Blocking, Finding Neighbors and Generating Candidates. IPM-Binary first builds an inverted index to store the mapping from a word to tuples containing it. Then tuples containing any word will be added to the blocking. We calculate the similarity between tuples in the blocking. Given a tuple t_i, we sort the tuples according to the similarity in descending order and choose top-k tuples as the neighbors. For the missing cell $t_i[A_k]$, IPM-Binary takes all values on attribute A_k from the neighbors $NN(t_i)$ as the candidates $can(t_i[A_k])$.

Step 4: Fine-tuning. Before training the model, we need to construct the training corpus. Complete tuples are considered positive training samples. For each positive sample, IPM-Binary will sample multiple negative samples. Then IPM-Binary fine-tunes the pre-trained language model to learn the ability of selecting the correct filling from the candidates.

Step 5: Imputing. With the fine-tuned model, IPM-Binary selects the most likely filling $\phi(t_i[A_k])$ from the candidates and impute the missing cell with it.

IV. Imputation with Pre-trained Language Models: Multi-Classifier

A straightforward idea is that we can consider the imputation problem as a multiclass classfication problem (IPM-Multi). As stated in Section III-A, IPM-Multi consists of 2 steps, fine-tuning and imputing.

A. *Fine-tuning in IPM-Multi*

IPM-Multi fine-tunes the pre-trained LM to learn how to select imputation from the domain $dom(A_k)$.

1) Pre-trained LM in IPM-Multi: Figure 3(a) is the architecture of the pre-trained model in IPM-Multi. Given the incomplete cell $t_i[A_k]$, we collect the values on complete attributes A_C^i in t_i as the input.

(1) The pre-trained LM first serializes the input attribute values into a sequence of tokens to conform to the input format of the pre-trained LM. A too complicated serializing strategy is unnecessary [4], [15]. Thereby, IPM-Multi simply serializes the inputs as

$$serialize(t_i[A_C^i]) = [CLS]\ \underbrace{T_1 \ldots}_{T_i \in t_i[A_p]}\ \cdots\ \underbrace{\ldots T_m}_{T_i \in t_i[A_q]}\ [SEP] \quad (2)$$

where $[CLS]$ and $[SEP]$ are the start and the end of the input, respectively, $A_p, A_q \in A_C^i$ are the complete attributes in t_i. The pre-trained LM transforms the input tokens into embeddings.

(2) The pre-trained LM captures the contextual information with the self-attention mechanism and outputs contextualized embeddings for the input embeddings.

(3) Following the existing works [7], [15], we take the contextualized embedding $E^c_{[CLS]}$ of token $[CLS]$ from the last layer as the summarization vector of the input. The reason is that $E^C_{[CLS]}$ is essentially a weighted average of all token representations, summarizing the rest of tokens. To summarize the input sequence, simply averaging all token representations could be suboptimal [12], since it will assign all tokens with the same weight including some tokens that are not relevant to the specific task. According to Formula 2, the first token of every sequence is always the special token $[CLS]$. The contextualized embedding $E^c_{[CLS]}$ of token $[CLS]$, computed using self-attention (Formula 1), is a weighted average of all token representations. Since the weights can be calculated for the specific task that we fine-tune on, $E^c_{[CLS]}$ can only collect the relevant information from the rest of the hidden states. Following the same line of [7], [15], which take $E^c_{[CLS]}$ as the summarization of input sequence and achieve good performance over various downstream tasks, we also take $E^c_{[CLS]}$ of token $[CLS]$ from the last hidden layer as the summarization vector of the input.

A multiclass classifier is then built. As shown in Figure 3(a), it takes $E^c_{[CLS]}$ as the input, and outputs the probability that each value in the domain is the correct filling. The multiclass classifier consists of linear layers and a softmax layer. It is the softmax layer that outputs the probability of each value being correct filling. The parameters of the multiclass classifier and the pre-trained LM will be updated together during fine-tuning.

In conclusion, we can see the IPM-Multi as a complex function $p^{(m)}(v, t_i, A_k)$ that calculates the probability that value v is the correct filling for $t_i[A_k]$.

2) Model Learning in IPM-Multi: If we impute the missing cells in all incomplete attributes simultaneously, the domain size will be the sum of the domain sizes of all incomplete attributes. Therefore, we choose to fine-tune the model for each incomplete attribute. Before fine-tuning the pre-trained LM, we need to build the training corpus first. Following the statement in Section IV-A1, the training example is $(t_i[A_C^i], l_i)$, where $l_i \in \{0, 1, ..., |dom(A_k)|-1\}$ is the class label of $t_i[A_k]$. We collect all the complete tuples and remove $t_i[A_k]$ from the tuples, which will be used as the ground truths. In this way, we can get the training corpus $T^{(m)} = \{(t_i[A_C^i], l_i)\}^{|N_c|}$, where $|N_c|$ is the amount of complete tuples.

The model is learned by minimizing the cross-entropy loss,

$$\mathcal{J}^{(m)} = -\sum_{i=1}^{|T^{(m)}|} \log(p_i^{(m)}[l_i]) \quad (3)$$

where $p_i^{(m)}[l_i]$ is the output probability of the class l_i.

Example 3 (Fine-tuning in IPM-Multi)**.** *Consider the running example in Table I. To impute more accurately the missing $t_1[A_{category}]$, we fine-tune the pre-trained LM on $A_{category}$.*

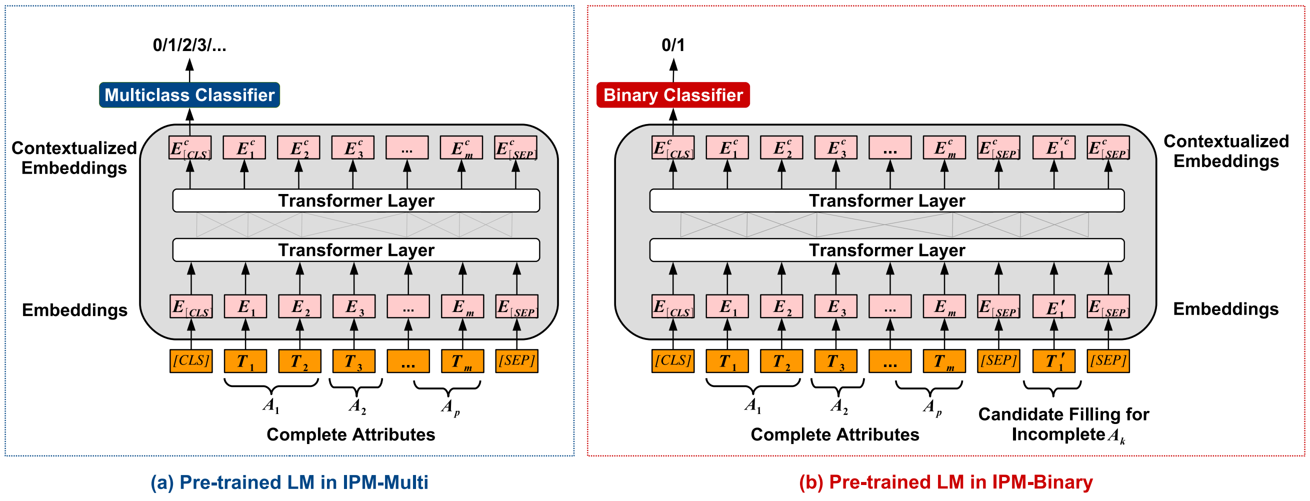

Fig. 3. Model Architecture. (1) The pre-trained language model first serialize the values of complete attributes and the possible filling into a sequence of tokens. Then the pre-trained LM transforms tokens into embeddings. (2) The stacked transformer layers then capture the hidden semantics between tokens and output the contextualized embeddings for each token. (3.a) A multiclass classifier takes the $E^c_{[CLS]}$ as the input and outputs the probability that each value in the domain is the correct filling for cell $t_i[A_k]$, as introduced in Section IV-A. (3.b) A binary classifier also takes the $E^c_{[CLS]}$ as the input and outputs the probability that the candidate filling is the correct filling for cell $t_i[A_k]$, as introduced in Section V-B.

Without introducing additional supervision, we utilize the complete tuples, such as t_3 whose $A_{category}$ is not missing, to fine-tune the model. The value of $t_3[A_{category}]$ is used as the label for training. The contents of t_3 are serialized as $[[CLS], belkin, sport, ..., ipod, nano, ..., 18.88, [SEP]]$. *We input the tokens into the pre-trained LM and obtain the summarization vector $E^c_{[CLS]}$ of the input. By taking $E^c_{[CLS]}$ as the input of the multiclass classifier in IPM-Multi, the classifier will output the probabilities of all classes being the imputation of $t_3[A_{category}]$. Assume that we have 4 classes in the active domain, and the 3rd class is the correct filling "mp3 accessories". That is, the ground truth of $t_3[A_{category}]$ is* 2 *(strating from 0). For example, if the output of the multiclass classifier is* $[0.1, 0.1, 0.75, 0.05]$, *the loss will be* 0.288. *The pre-trained LM and the multiclass classifier will be updated together via backpropagation.*

B. Imputing in IPM-Multi

Following the problem setting in Section II-B, we select the value with the highest probability, i.e.,

$$\phi(t_i[A_k]) = \underset{v \in dom(t_i[A_k])}{argmax} \; p^{(m)}(v, t_i, A_k) \tag{4}$$

where $p^{(m)}(v, t_i, A_k)$ denotes the output probability of IPM-Multi on the class whose value is v. The pseudocode of IPM-Multi is shown in Appendix IX-A in [1].

Example 4 (Imputing in IPM-Multi)**.** *Similar to the input of t_3 in the training phase in Example 3, we also input the complete values in t_1 to the fine-tuned model. Assume that the output of the fine-tuned model is* $[0.1, 0.05, 0.8, 0.05]$. *IPM-Multi would impute $t_1[A_{category}]$ with the 3rd class, i,e, "mp3 accessories", referring to the highest 0.8.*

V. Imputation with Pre-trained Language Models: Binary Classifier

Owing to the low redundancy of databases and large domain sizes, training data are not always sufficient. For example, the domain size of the "brand" attribute in Flipkart dataset is 3453, as shown in Table II. However, there are only 8884 tuples in Flipkart dataset. It means that only about 2 tuples are corresponding to each class. Obviously, this is not enough to train an accurate multiclass classifier for IPM-Multi. In this case, IPM-Multi will fall into the over-fitting problem.

In order to solve the problem, we propose an approach named IPM-Binary, which selects imputation from neighbors with pre-trained LMs. IPM-Binary first generates possible fillings $can(t_i[A_k])$ for each missing cell $t_i[A_k]$. Then we fine-tune a pre-trained language model to select the value $v \in can(t_i[A_k])$ with the highest probability $p(v, t_i, A_k)$.

It is worth noting that IPM-Binary does not highly depend on redundant tuples, thanks to the abundant co-occurrence semantics w.r.t. tokens captured by the pre-trained models and the attention mechanism. By considering the implicit semantics of tokens extracted in the pre-trained LM and explicit co-occurrence of tokens in candidates and complete values, IPM-Binary learns further the semantical relationships between tokens in the fine-tuning process.

Furthermore, a set of possible fillings $can(t_i[A_k])$ significantly reduce the search space of imputation compared to $dom(A_k)$ in IPM-Multi. The overview is in Figure 2. We will introduce each step in detail. Section V-A contains Step 1-3. Step 4 is in Section V-B, and Step 5 is in Section V-C.

A. Candidates Generation in IPM-Binary

There are many existing methods [8], [31], [39] that can generate a set of possible values for imputation. A common approach is to generate candidates from neighbors. Similar to many Entity Resolution works [9], finding neighbors needs to compare pairs of tuples. The time cost of comparison will be very high, if we have to compare each pair in a large dataset. Assume that there exist 1000 tuples in the dataset. We need to compare around 1000000 tuple pairs, which is unacceptable. Therefore, the first step in IPM-Binary is blocking, which could significantly reduce the number of comparisons.

Since IPM-Binary applies a pre-trained language model, we need to focus on the words in the tuple. Thereby, we find neighbors according to the word co-occurrence relationship.

First, IPM-Binary will build an inverted index $Index(R)$, where the key is the word w and the value are the set of tuples containing the word $\{t|w \in t\}$. Tuples that exist in the same block $Index(R)[w]$ will be considered as neighbors. To avoid large blocks, we ignore the word whose frequency is larger than the given threshold τ, i.e., $W_\tau = \{w|Index(R)[w]| \leq \tau\}$. Then, we can obtain the block:

$$C = \bigcup_{w \in W_\tau} \{(t_i, t_j) t_i, t_j \in |Index(R)[w]\}$$

Second, IPM-Binary will calculate the similarity between each pair of tuples $(t_i, t_j) \in C$. Following the principle that the atomic units are tokens, we choose term jaccard similarity as the similarity metric,

$$Sim(t_i, t_j) = \frac{|S(t_i) \bigcap S(t_j)|}{|S(t_i) \bigcup S(t_j)|} \tag{5}$$

where $S(t_i)$ denotes the set of tokens in tuple t_i.

Intuitively, the higher the similarity of the pair of tuples is, the more likely they can fill each other. Thus, we sort tuples according to their similarities in descending order and take the top-k tuples as neighbors, i.e.,

$$NN(t_i) = \{t_j | (t_i, t_j) \in C\}^\kappa$$

Note that a large κ will increase the time cost for candidate selection, while a small κ may cause IPM-Binary to miss the correct fillings. If the computing resources are sufficient, we recommend setting κ with a large number to ensure a good imputation result. Experiment results about the impact of κ is in Appendix IX-E in [1].

Third, IPM-Binary will generate the possible fixes for the missing cell $t_i[A_k]$ from its neighbors. We take the value of all neighbors $NN(t_i)$ on the attribute $[A_k]$ as the candidates,

$$can(t_i[A_k]) = \{t_j[A_k] | t_j \in NN(t_i)\}$$

B. Fine-tuning in IPM-Binary

1) Pre-trained LM in IPM-Binary: Deep learning based methods, such as IPM-Multi in Section IV, naturally model the imputation problem as a multiclass classfication problem, where the number of classes is the domain size $|dom(A_k)|$. Due to the low redundancy of databases, the number of classes is large while training samples are relatively limited. Hence, if we directly train a multiclass classifier for imputation, the model will likely fall into the over-fitting pitfall. For this reason, we generate a set of possible fillings $can(t_i[A_k]) \subseteq dom(A_k)$ for each missing cell $t_i[A_k]$. Compared to the domain $dom(A_k)$, the search space of $can(t_i[A_k])$ is significantly smaller.

Considering the above issues, we formulate the imputation task as a binary decision. Motivated by Next Sentence Prediction (NSP) task in BERT [7], we utilize the pre-trained LM to identify whether the candidate value is a correct filling.

(1) As Figure 3(b) illustrated, IPM-Binary also needs to serialize the attribute values in the tuple into a sequence of tokens. Unlike IPM-Multi, IPM-Binary takes the combination of complete attribute values as the first "sentence" and the candidate filling for the incomplete attribute as the second "sentence".

$$\begin{aligned} & serialize(t_i[A_C^i], t_i[A_k]) \\ & = [CLS] \underbrace{T_1 \ldots T_p}_{T_i \in t_i[A_{c1}]} \ldots \underbrace{T_q \ldots T_m}_{T_i \in t_i[A_{cx}]} [SEP] \underbrace{T_1^{'} \ldots T_l^{'}}_{T_i \in v_k} [SEP] \end{aligned}$$

where A_{cx} denotes the complete attribute in the tuple t_i, $v_k \in can(t_i[A_k])$ denotes the candidate filling. The pre-trained LM then transforms the input tokens into embeddings.

(2) Similar to IPM-Multi, IPM-Binary also takes the contextualized embedding $E^c_{[CLS]}$ of token $[CLS]$ from the last layer as the summarization vector of the input.

(3) As illustrated in Figure 3(b), a binary classifier is built by taking $E^c_{[CLS]}$ as the input. The binary classifier is used to decide whether the candidate value $v_k \in can(t_i[A_k])$ is a correct filling for the missing cell $t_i[A_k]$. The parameters of the binary classifier and the pre-trained LM will be updated together during fine-tuning.

In conclusion, we can see the pre-trained language model as a complex function $p^{(b)}(v_k, t_i, A_k)$. It outputs the probability that the candidate value fills the incomplete tuple.

2) Model Learning in IPM-Binary: Given a training corpus $T^{(b)} = \{(t_i[A_C^i], v_k, y_n)\}_{n=1}^{|T|}$, where $t_i[A_C^i]$ is the combination of complete attribute values, v_k denotes the candidate filling for cell $t_i[A_k]$, and $y_n \in \{0, 1\}$ indicates whether v_k is the correct filling for $t_i[A_k]$. We train our imputation model by minimizing the cross-entropy objective function,

$$\mathcal{J}^{(b)} = -\sum_{n=1}^{|T^{(b)}|} [y_n \log(\hat{p}_n) + (1 - y_n) \log(1 - \hat{p}_n)] \tag{6}$$

where $\hat{p}_n$ is the output of the classifier.

3) Negative Sampling: Considering that there exist differences in semantic relationships between different attributes. We fine-tune the pre-trained language model individually for each missing attribute. Naturally, IPM-Binary needs to build an individual training corpus $T^{(b)}$ for each attribute A_k.

For positive training samples $T_p^{(b)}$, we select all tuples whose attribute A_k are not missing,

$$T_p^{(b)} = \{(t_i[A_C^i \backslash A_k], t_i[A_k], 1)\}.$$

To train a binary classifier, we need negative sampling to obtain a balanced training set of positive and negative samples. If we take all values not equal to $t_i[A_k]$ as the negative samples of imputation, the training corpus will be highly imbalanced with a few positive and a large number of negative samples. This will make the classifier biased towards negative class and increase the time cost of training.

Following [34], IPM-Binary also samples η negative samples for each positive sample. η is a hyper-parameter named negative sampling ratio. An intuitive idea is to sample values not equal to the existed cell value from the domain. Formally, for the tuple t_i whose positive sample is $(t_i[A_C^i \backslash A_k], t_i[A_k], 1)$, we sample its negative samples $T_n^{(b)}[t_i]$ from the domain.

$$T_n^{(b)}[t_i] = \{(t_i[A_C^i \backslash A_k], v_n, 0)\}^{\eta}$$
$$v_n \neq t_i[A_k], v_n \in dom(A_k)$$

Values not equal to $t_i[A_k]$ in $dom(A_k)$ are sampled uniformly.

However, most of the values in the domain are not semantically related to the tuple in the positive sample. If we conduct negative sampling over the domain, the negative samples might be easy for the pre-trained language model to distinguish. Thus, fine-tuning will be a trivial task, so that the fine-tuned model cannot select the correct filling from several semantically related candidates.

To resolve this, we propose to conduct negative sampling over the neighbors $NN(t_i)$ rather than the domain $dom(A_k)$.

$$T_n^{(b)}[t_i] = \{(t_i[A_C^i \backslash A_k], v_n^{'}, 0)\}^{\eta}$$
$$v_n^{'} \neq t_i[A_k], v_n^{'} \in can(t_i[A_k])$$

where all values not equal to $t_i[A_k]$ in $can(t_i[A_k]) = \{t_j[A_k] | t_j \in NN(t_i)\}$ are sampled uniformly. Recall that each pair of neighbors share at least one token. It means that $v_n^{'}$ and $t_i[A_k]$ are probably semantically related. In this way, we generate a more challenging training corpus, where the candidate values in the negative samples are likely to be semantically related to the positive sample. During fine-tuning on the training corpus, the model will be forced to learn to find the correct match from several candidates that are semantically related to the tuple. Experiment results verify the effectiveness of this sampling strategy in Appendix IX-F in [1].

Example 5 (Fine-tuning in IPM-Binary)**.** *Consider again the running example in Table I with missing $t_1[A_{category}]$. We also fine-tune IPM-Binary on the complete tuples, such as t_3, whose $A_{category}$ are not missing. First, we need to generate the training corpus. The positive sample of t_3 is $(t_3[A \setminus A_{category}], mp3\ accessories, 1)$. Given negative sampling ratio $\eta = 1$, we find one negative candidate "electronics general". The negative sample of t_3 is thus $(t_3[A \setminus A_{category}], electronics\ general, 0)$. We serialize the training samples and input them into the model for fine-tuning. For instance, we serialize the positive training sample as $[[CLS], belkin, ..., 18.88, [SEP], mp3\ \ accessories, [SEP]]$. IPM-Binary inputs the token series into the pre-trained LM and obtain the summarization vector $E_{[CLS]}^c$ of the input. By taking $E_{[CLS]}^c$ as the input of the binary classifier in IPM-Binary, the classifier outputs the probability that the candidate is correct. For instance, if the binary classifier outputs 0.8 for "mp3 accessories" and 0.6 for "electronics general", the mean loss will be 0.570. The pre-trained LM and the binary classifier will be updated together via backpropagation.*

C. Imputing in IPM-Binary

And then, IPM-Binary will select one candidate value as the final filling $\phi(t_i[A_k])$ for each missing cell $t_i[A_k]$ with the fine-tuned pre-trained language model. As stated in Section V-B, the fine-tuned pre-trained LM could be abstracted as a function $p^{(b)}(v, t_i, A_k)$. The output of the function is the probability that the candidate value v is the correct filling for the missing cell t_i, A_k, i.e.,

$$p^{(b)}(v, t_i, A_k) = P(\phi^*(t_i[A_k]) = v)$$

where $\phi^*(t_i[A_k])$ is the true filling.

To reduce the search space, we only select the candidate filling v from the neighbors $v \in \{t_j[A_k] | t_j \in NN(t_i)\}$. Intuitively, we tend to select the candidate filling with the highest probability, i.e.,

$$\phi(t_i[A_k]) = \underset{v \in can(t_i[A_k])}{argmax}\ p^{(b)}(v, t_i, A_k) \tag{7}$$

However, we cannot make sure that the candidate $can(t_i[A_k])$ must contain the correct filling. In this case, if we always choose top-1 candidate as the filling, we will obtain a wrong filling result. For many downstream tasks such as Entity Resolution, wrong values will be more harmful than missing values. For this reason, we suggest not to impute the missing cell $t_i[A_k]$ whose candidate $can(t_i[A_k])$ does not contain any value with high confidence.

We can filter candidates with relatively low confidence by setting a threshold ϵ.

$$can^{(\epsilon)}(t_i[A_k]) = \{v | v \in can(t_i[A_k]), p^{(b)}(v, t_i, A_k) \geq \epsilon\}$$

Note that if we set $\epsilon = 0$, this is equivalent to the "top-1" selection strategy. If the missing cell $t_i[A_k]$ whose candidate $can(t_i[A_k])$ does not contain any value with $p(v, t_i, A_k) \geq \epsilon$, the filtered candidate will be a empty set, i.e., $can^{(\epsilon)}(t_i[A_k]) = \emptyset$. It means that IPM-Binary would not impute it. In this way, we can avoid introducing too many errors during imputation. Experiment results about the impact of this selection threshold is in Appendix IX-G in [1]. The pseudocode of IPM-Binary is shown in Appendix IX-B in [1].

Example 6 (Imputing in IPM-Binary)**.** *Assume that we obtain two candidates, "mp3 accessories" and "electronics general", for $t_1[A_{category}]$ in Table I. Similar to the samples of t_3 in the training phase in Example 5, we also input the combined tuples of complete values and the candidates of t_1 to the fine-tuned model. If the fine-tuned model outputs 0.9 for "mp3 accessories" and 0.4 for "electronics general", then IPM-Binary will impute $t_1[A_{category}]$ with the "mp3 accessories".*

TABLE II
DATASET SUMMARY

Dataset	Size	# Attributes	# Incomplete Attributes	Avg Domain Size
Restaurant[2]	864	4	1	49
Walmart[3]	2300	5	2	251
Amazon[3]	13797	4	2	939
Buy[4]	651	3	1	62
Housing[5]	5000	11	10	966
Phone[6]	4312	6	1	335
Zomato[7]	50279	10	1	93
Flipkart[8]	8884	9	1	3453

VI. EXPERIMENTS

A. Experimental Settings

The experiments run on a machine with 2.1GHz CPU and Nvidia 1080ti GPU. RoBERTa [1] is the default pre-trained LM.

1) Datasets: We employ several real datasets in experiments. Table II provides statistics on data sizes (the number of tuples), schema sizes (the number of attributes) and average domain sizes of incomplete attributes. Such information can be used as a justification for the choices of classifiers, IPM-Multi or IPM-Binary, as discussed in Section VI-C1.

In particular, the Zomato dataset contains a large number of 50279 tuples, while dataset Housing has 11 attributes.

The Phone dataset contains a number of 656 naturally occurring missing values in attribute "brand". We manually label the corresponding truths of missing values for evaluation.

For all datasets except Phone, we randomly remove values from several categorical attributes. The missing rates are applied on per incomplete attribute. For instance, if we need to remove 10% values from the dataset Walmart, then we will randomly remove 10% attribute values from the two categorical attributes "category" and "brand", respectively. (Our proposal is robust to the missing rate as illustrated in Appendix IX-D in [1].)

It is worth noting that imputation is not always possible, since the randomly removed values may no longer appear in the active domain of the attribute at hand. For such missing values not in the active domain, our proposal IPM (as well as HoloClean [38]) that rely on value redundancy fail to impute.

2) Criteria: We measure the accuracy on whether the imputation results are equal to the ground truth. $truth$ denotes the set of truth values and imp denotes the imputed values. The accuracy acc is calculated by $acc = \frac{|truth \cap imp|}{|truth \cup imp|}$, i.e., the proportion of missing values accurately imputed. For datasets with several missing attributes, we report the mean accuracy of the missing attributes. To avoid randomness, all the experiments were repeated five times. The average of the five results will be used as the final results. To demonstrate that the differences in algorithms are real in all the experiments, we conduct statistical significance calculations by Student's Paired t-test [30]. The best performances based on Student's Paired t-test at 95% significance level (i.e., $p < 0.05$) are bold in Tables III, IV and V.

3) Baselines: We select the state-of-the-art methods in each category of imputation methods as summarized in Section VII of related work. For nearest-neighbor based methods, we select kNNE [8] and MIBOS [39]. For clustering based methods, we use CMI [42] based on the pre-trained embeddings [16]. For statistics based methods, we choose ERACER [19] and HoloClean [27], [38]. We report the results of the latest version of HoloClean, i.e., attention-based HoloClean [38]. These methods are all applicable for categorical data imputation and often compared in the existing studies [31], [32]. We also adapt the error correction method Baran [17] to "correct" the null values. Following the default setting of Baran [9], we set the number of labels as 20. In addition, we compare with several generative model based approaches MIDAS [10], MIWAE [18], HI-VAE [21], DataWig [3] on numerical attributes. (Please see implementation details in Appendix IX-C in [1].)

B. Comparison with Existing Approaches

1) Comparison over Categorical Dataset: Table III reports the accuracy performance of the implemented baselines and our proposal IPM-Multi and IPM-Binary. The column (#Train / #Total) presents how much training data was used. Among the training set, 80% data are used for training and 20% for validation to avoid over-fitting.

As we can see from the Table III, our proposed methods benefit from the ability to capture semantic information in data and outperform baselines over all datasets. Some baselines may fail to correctly impute any value with accuracy 0. Similarities between long texts in data cannot be accurately evaluated by symbolic similarity functions applied in the baselines. CMI with embeddings is still weak, since it is not fine-tuned for the imputation task as our proposal.

The results in Table III show that our proposal outperforms HoloClean, especially over the datasets with text attributes, such as "title" and "description", where the statistics fail to work. On the dataset Zomato with redundant tuples, HoloClean achieves good performance, while IPM can also benefit from redundancy and still outperform HoloClean. Rather than the limited co-occurrence at the attribute value level in HoloClean, our IPM utilizes the abundant co-occurrence semantics w.r.t. tokens. Thus, the accuracy of IPM on the dataset with low redundancy such as Amazon is higher than HoloClean.

As aforesaid, the error correction method Baran [17] is adapted to "correct" the null values. Unfortunately, in the imputation problem, the original value is missing (null value). Since there is no original error value to use, Baran cannot learn

[1] provided by Hugging Face: https://huggingface.co/

[2] https://www.cs.utexas.edu/users/ml/riddle/data.html

[3] https://sites.google.com/site/anhaidgroup/useful-stuff/data

[4] https://dbs.uni-leipzig.de/en

[5] https://www.kaggle.com/harlfoxem/housesalesprediction

[6] https://www.kaggle.com/PromptCloudHQ/amazon-reviews-unlocked-mobile-phones

[7] https://www.kaggle.com/himanshupoddar/zomato-bangalore-restaurants

[8] https://www.kaggle.com/PromptCloudHQ/flipkart-products

[9] https://github.com/BigDaMa/raha/tree/master/raha

TABLE III
IMPUTATION ACCURACY OF IPM COMPARED TO THE EXISTING APPROACHES (MISSING RATE = 10%, THE BEST PERFORMANCES BASED ON STUDENT'S PAIRED T-TESTS AT 95% SIGNIFICANCE LEVEL ARE BOLD)

Dataset	(#Train / #Total)	kNNE	MIBOS	CMI	ERACER	DLM	Baran	HoloClean	IPM-Multi	IPM-Binary
Restaurant	623 / 864	0.206 ± 0.019	0.093 ± 0.025	0.560 ± 0.049	0.185 ± 0.031	0.631 ± 0.024	0.315 ± 0.044	0.331 ± 0.056	$\mathbf{0.772 \pm 0.023}$	0.660 ± 0.008
Walmart	1660 / 2300	0.207 ± 0.019	0.142 ± 0.097	0.571 ± 0.014	0.032 ± 0.000	0.641 ± 0.005	0.173 ± 0.071	0.247 ± 0.017	0.703 ± 0.039	$\mathbf{0.822 \pm 0.015}$
Amazon	9943 / 13797	0.128 ± 0.005	0.007 ± 0.001	0.533 ± 0.006	0.001 ± 0.000	0.636 ± 0.005	0.037 ± 0.004	0.095 ± 0.008	$\mathbf{0.725 \pm 0.005}$	0.686 ± 0.005
Buy	469 / 651	0.355 ± 0.016	0.016 ± 0.011	0.653 ± 0.029	0.003 ± 0.006	0.709 ± 0.025	0.227 ± 0.173	0.162 ± 0.043	0.637 ± 0.039	$\mathbf{0.965 \pm 0.015}$
Zomato	36217 / 50279	0.581 ± 0.005	0.024 ± 0.001	0.746 ± 0.006	0.001 ± 0.000	OUT OF MEMORY	0.938 ± 0.023	0.956 ± 0.005	$\mathbf{0.995 \pm 0.001}$	0.976 ± 0.001
Phone	2924 / 4312	0.466 ± 0.000	0.000 ± 0.000	0.404 ± 0.000	0.175 ± 0.000	0.619 ± 0.000	0.162 ± 0.073	0.130 ± 0.055	0.847 ± 0.005	$\mathbf{0.867 \pm 0.002}$
Flipkart	6408 / 8884	0.219 ± 0.007	0.000 ± 0.000	0.463 ± 0.024	0.000 ± 0.000	0.524 ± 0.023	0.353 ± 0.151	0.105 ± 0.033	0.206 ± 0.008	$\mathbf{0.680 \pm 0.016}$

the value-based model to determine corrections. Therefore, the performance of Baran is worse than IPM as illustrated in Table III.

The Phone dataset has a number of 656 naturally occurring missing values in attribute "brand", with manually labeled truths, as introduced in Section VI-A1. The results of naturally occurring missing values are similar to the synthetic ones in Table III. Our proposal IPM shows an accuracy improvement of 0.228 compared to the next-best method DLM.

On the dataset with a higher number of classes, IPM-Binary still achieves good performance, while the accuracy of IPM-Multi depends on the number of training samples in each class. For instance, on the dataset Flipkart with 3453 classes and only 2 training samples per class, IPM-Binary still achieves a high accuracy 0.68, while the accuracy of IPM-Multi is only 0.21. On the other hand, IPM-Multi can also achieve good results on the dataset such as Amazon with many classes but abundant training samples per class. It verifies the intuition that IPM-Multi relies on the abundant examples in each class for training. Instead, thanks to the co-occurrence semantics w.r.t tokens in the candidates and complete attributes, IPM-Binary can achieve a better performance with less training data.

A large data size and schemata will not be an obstacle to IPM, but more conducive to IPM. Given a larger dataset, IPM will achieve better performances with more training data. And IPM also benefits from more co-occurrences w.r.t tokens in datasets with a large schemata. As reported in Table III, our proposal can still achieve good performances on large datasets such as Zomato with 50279 tuples and 10 attributes.

2) Comparison over Mixed-type Dataset: We conduct experiments on more challenging dataset Housing, including 4 categorical attributes and 6 numerical attributes, in Table IV. To handle numerical attributes, we employ the generative model based methods, MIDAS [10], MIWAE [18], HI-VAE [21], DataWig [3], as introduced in Section VII-D. Again, the missing rate is 10%. It is not surprising that our proposal IPM shows higher imputation accuracy on categorical attributes, referring to the results in Section VI-B1.

Interestingly, IPM also has lower imputation NRMS (normalized Root Mean Square error) on numerical attributes, compared to the generative model based approaches MIDAS [10] and HI-VAE [21], since we can handle some numerical values. Our solution based on pre-trained language models can capture some semantics on the numerical values, such as between value "4" in attribute "bed" and value "four" in attribute "description" in dataset Housing. However, IPM cannot

TABLE IV
MIXED TYPE DATA (THE BEST PERFORMANCES BASED ON STUDENT'S PAIRED T-TESTS AT 95% SIGNIFICANCE LEVEL ARE BOLD)

Methods	Categorical(Accuracy)	Numerical(NRMS)
MIDAS	0.284 ± 0.009	0.173 ± 0.019
MIWAE	0.222 ± 0.016	0.095 ± 0.005
HI-VAE	0.368 ± 0.017	0.196 ± 0.053
DataWig	0.190 ± 0.010	0.095 ± 0.011
HoloClean	0.586 ± 0.011	0.076 ± 0.006
IPM-Binary	$\mathbf{0.612 \pm 0.040}$	0.159 ± 0.008
IPM-Multi	$\mathbf{0.660 \pm 0.019}$	0.134 ± 0.003
IPM-Binary+HoloClean	$\mathbf{0.612 \pm 0.040}$	$\mathbf{0.067 \pm 0.007}$
IPM-Multi+HoloClean	$\mathbf{0.660 \pm 0.019}$	$\mathbf{0.067 \pm 0.006}$

handle the numerical attributes that basically do not contain redundant values, such as "price" and "latitude". Nevertheless, our proposal is complementary to the existing methods, e.g., IPM+HoloClean by applying IPM first on categorical attributes and then HoloClean for numerical attributes. As shown, IPM-Multi+HoloClean achieves the best performance on the dataset Housing mixed with categorical and numerical attributes.

C. Evaluation of Proposed Methods

1) Varying Amount of Training Data: We also test IPM-Binary and IPM-Multi with different amounts of training data. Recall that the negative samples are proportional to the positive samples. Thus, we change the amounts of training data by adjusting the number of positive training samples.

The results over four datasets are reported in Figure 4. Both IPM-Binary and IPM-Multi achieve better performances with more data. As shown in Figure 4(c), we find IPM-Binary outperforms IPM-Multi when we only use 10% training data. As the size of training data increases, IPM-Multi gradually catches up with IPM-Binary as illustrated in Figure 4(g).

If there is sufficient training data, IPM-Multi will achieve better results as stated in Section VI-B1. Otherwise, the results of IPM-Binary will be better as shown in Figure 4(a) and 4(e). By "sufficient" we mean the number of training samples in each class. As reported in Figure 4(h), the average number of samples in each class is only about 10 in the Amazon dataset with 90% training data. In contrast, IPM-Multi shows better results in dataset Restaurant which has more samples in each class as shown in Figure 4(d). It verifies the intuition that IPM-Multi relies on abundant training samples in each class.

It is worth noting that IPM is a self-supervised method, where the set of complete tuples are used as the training data. Given a new dataset, if there exist more than 10 training samples in each class, i.e., $\frac{\text{Data Size}}{\text{Domain Size}} \geq 10$ according to the

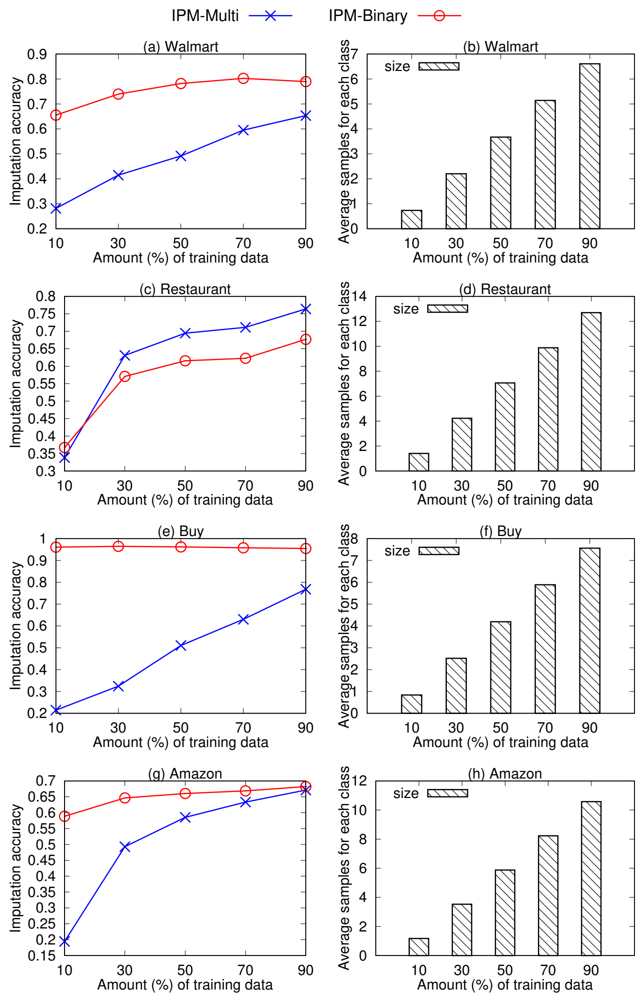

Fig. 4. Varying Amount (%) of Training Data

statistics in Table II, IPM-Multi may have better performance than IPM-Binary in practice as empirically shown in Figure 4.

We can find that IPM-Binary can achieve good performance with only 10% training data. This is because there exists a pattern in the Buy dataset. The pattern is that the first few words in the tuple generally denote the brand. IPM-Binary can capture the semantic relation by inputting both values of complete attributes and candidate values. However, IPM-Multi just inputs the values of complete attributes and treats all values in the domain as labels without semantics. Thus, IPM-Multi fails to find the semantic relationship, which is useful in imputation over the Buy dataset.

2) Varying Data Redundancy: To inspect the impact of redundancy in the data, we conduct the experiment on varying the number of redundant tuples in dataset Zomato. For instance, a redundancy rate 3 denotes that 3 version tuples of the same entity are included in the test. As illustrated in Figure 5(a), our proposal is not highly dependent on redundant tuples. Thanks to the abundant co-occurrence semantics w.r.t. tokens captured by the pre-trained models and the attention

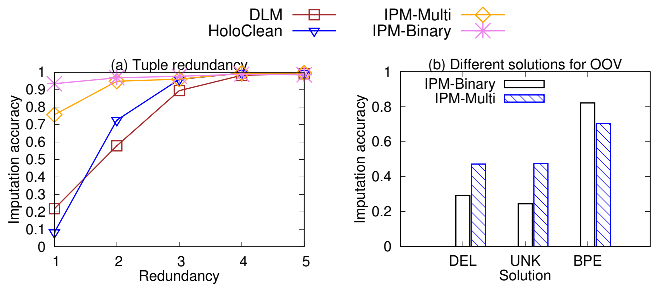

Fig. 5. Experimental results of (a) Varying data redundancy and (b) Accuracy of different solutions for OOV

TABLE V
IMPUTATION ACCURACY OF IPM WITH DIFFERENT PRE-TRAINING TECHNIQUES (MISSING RATE = 10%, THE BEST PERFORMANCES BASED ON STUDENT'S PAIRED T-TESTS AT 95% SIGNIFICANCE LEVEL ARE BOLD)

Dataset	IPM-Multi (DistilBERT)	IPM-Binary (DistilBERT)	IPM-Multi (RoBERTa)	IPM-Binary (RoBERTa)
Restaurant	$\mathbf{0.769 \pm 0.038}$	0.711 ± 0.034	$\mathbf{0.772 \pm 0.023}$	0.660 ± 0.008
Walmart	0.676 ± 0.026	$\mathbf{0.821 \pm 0.021}$	0.703 ± 0.039	$\mathbf{0.822 \pm 0.015}$
Amazon	0.667 ± 0.002	0.683 ± 0.008	$\mathbf{0.725 \pm 0.005}$	0.686 ± 0.005
Buy	0.580 ± 0.033	$\mathbf{0.964 \pm 0.021}$	0.637 ± 0.039	$\mathbf{0.965 \pm 0.015}$

mechanism, our proposal achieves high accuracy even without any redundant tuple (redundancy=1).

3) The Impact of Pre-training: To understand the impact of pre-training, we compare the performance of IPM using different pre-training techniques, DistilBERT [28] and RoBERTa [16]. DistilBERT is a distilled version of BERT, pre-trained on BooksCorpus (800M words) [43] and English Wikipedia (2,500M words) [7]. RoBERTa is pre-trained on five English corpus (BooksCorpus [43], Wikipedia [7], Ccnews[10], OpenWebText[11], Stories [35]), 160GB uncompressed text.

Compared to DistilBERT, RoBERTa is a larger LM pre-trained on a larger corpus and achieves better performances on various NLP tasks as reported in [16], [28]. Therefore, by capturing more and better semantics, the accuracy of IPM(RoBERTa) is slightly higher than IPM(DistilBERT) as reported in Table V. In short, the pre-training technique showing better performance in the original NLP tasks leads to better imputation results as well.

4) The Impact of Out-of-vocabulary Words: To handle the values that are enterprise-specific and out of the vocabulary (OOV) of the pre-trained models, we employ Byte-Pair Encoding (BPE) [25], [29]. BPE is a hybrid between character- and word-level representations, often used for handling large vocabularies [7], [16], [28].

As illustrated in Figure 5(b), compared to deleting (DEL) and replacing the OOV words with the special token $< unk >$ (UNK), IPM with BPE achieves a significantly higher accuracy. Recall that IPM-Binary relies on the co-occurrence semantics w.r.t. tokens. By considering OOV words with BPE, IPM-Binary can utilize more co-occurrence w.r.t tokens, and thus improve the performance sharply.

[10]http://commoncrawl.org/2016/10/newsdataset-available
[11]https://skylion007.github.io/OpenWebTextCorpus/

VII. Related Work

A. Imputation based on Neighbors

The intuition to impute missing values is to retrieve the complete values and find similar neighbors for imputation. The k-nearest-neighbor approach (kNN) [2] tries to find the nearest neighbors of a tuple t_i with the smallest distance on complete attributes $t_i[A_C^i]$, and then utilizes the complete values on missing attributes $t_i[R \backslash A_C^i]$ to impute. The kNN Ensemble (kNNE) [8], a variation of kNN, then enhances kNN by retrieving more neighbors over various subsets and then combines the results. Moreover, MIBOS [39] designs another criterion for similarity according to the undifferentiated set of tuples, which is actually the number of the consistent attributes between tuples. However, the limitation is that the neighbor-based approaches generate imputation candidates only according to given similarity functions, whereas the similarity functions are not always effective on target datasets and not capable to deal with semantical similarity.

B. Imputation based on Clustering

The clustering-based imputation methods employ clustering techniques to divide tuples into groups and then perform imputation on groups respectively. For instance, fuzzy k-means [14] and GMM [40] are applied to provide clustering results for imputation on numerical data. CMI [42] adopts k-means and g-means first and then applies kernel function imputation strategy to impute categorical data. Clustering-based imputation methods still suffer from the aforementioned limitation due to the same similarity strategies for imputation.

C. Imputation based on Statistical Models

The imputation methods based on statistical models learn from the correlated values of different attributes, both within and across tuples (e.g., ERACER [19]). After building the models, statistical inferences (i.e., candidates) could be generated. And then the candidate that maximizes the likelihood of the probabilistic models will be selected to fill the missing entries. Different from likelihood on attributes, distance likelihood maximization (DLM) [31] proposes to consider the likelihood on distances, which outperforms existing methods both on categorical and numerical data.

HoloClean [27], [38] performs statistical inference on the factor graph generated by rules. It also utilizes the attention mechanism to learn structural properties of the data distribution and impute missing values with the learned structure. The differences between HoloClean and our proposal IPM are as follows. (1) The granularity of co-occurrence is different. HoloClean considers each attribute value as a whole and studies the co-occurrence of different attribute values, while our IPM utilizes the co-occurrence of more fine-grained tokens in attribute values, such as between "hdmi cables" and "tv accessories". Thanks to the pre-trained language models, such co-occurrence semantics may not necessarily appear in the dataset but have already been captured over the huge corpus where the language models are pre-trained. (2) The attention mechanism w.r.t. co-occurrence is different. Due to lack of co-occurrences of attribute values, HoloClean [38] compromises to learn the attention weights that are only dependent on attributes rather than values. In contrast, thanks to the more abundant co-occurrences w.r.t. tokens in the dataset, IPM is able to employ the attention mechanism, at a finer granularity, to capture further semantics. In summary, rather than the limited co-occurrence at the attribute value level in HoloClean, IPM utilizes the abundant co-occurrence semantics w.r.t. tokens that are (1) implicitly extracted in the pre-trained language models and (2) explicitly exist in the dataset at hand.

D. Imputation based on Generative Models

The imputation methods based on generative models employ deep generative models to learn latent distribution from the observations. For instance, MIDAS [10] employs a class of denoising autoencoders to perform multiple imputation. MIWAE [18] leverages importance-weighted autoencoder (IWAE), a kind of deep latent variable model, to learn the latent distribution of the data. In HI-VAE [21], variational autoencoders are utilized to capture the latent structure and then solve the imputation tasks. By combining deep learning based feature extractors with automatic hyperparameter tuning, DataWig [3] provides an end-to-end framework to impute missing values. While these methods naturally apply to numerical data, to deal with the categorical data, they have to use one-hot encoding, i.e., fail to capture semantics in words. In contrast, our IPM benefits from the semantics, and thus outperforms these methods on imputing categorical data.

E. Error Correction based on Classification

Baran [17] transforms the error correction problem from a multiclass classification task to binary classification. The differences between Baran and our IPM-Binary are as follows. (1) The problem input is different. The error correction problem studied by Baran needs a pair of error and its correction as the input, while the imputation problem considered in this study has no original value (missing) in the input. (2) The pre-training mechanism is different. Baran pre-trains models based on Wikipedia revisions of error and correction pairs. Since there is no original value available in imputation, we employ the models pre-trained on various corpus such as BooksCorpus [43] and Wikipedia [7] to capture token semantics.

It is worth noting that Baran [17] relies on external information to handle out of domain corrections. Baran collects out of domain candidates by mapping the original error values to the revision history, e.g., in Wikipedia. Such original error values, however, are not available in the imputation problem. We leave this challenging study of leveraging external sources for imputation as future work.

VIII. Conclusion

In this paper, to determine a better filling for the missing cell in relational data, we propose to take advantage of semantics to improve the accuracy. The existing data imputation methods determine the filling via symbolic similarity, which is not effective for data containing long texts as illustrated in

Example 1. For this reason, our proposal IPM, consisting of IPM-Multi and IPM-Binary, is the first work of imputing data with semantics. To reduce the requirement of training data, we use pre-trained LMs to capture semantics. A straightforward solution is to input the tuple as a sequence of words and predict the missing values via a multiclass classifier (IPM-Multi). IPM-Multi achieves good performance with sufficient training data. However, this condition is difficult to meet due to the low redundancy of databases and large domain sizes. IPM-Multi may fall into the pitfall of over-fitting. To solve this problem, we propose IPM-Binary, which identifies whether the possible fillings in the candidates is correct via a binary classifier. Compared to IPM-Multi, IPM-Binary requires less training data. The comparisons with existing approaches demonstrate that semantics are significant for imputation. The extensive experiments verify the rationality of our proposal.

Acknowledgment

This work is supported in part by the National Key Research and Development Plan (2019YFB1705301), the National Natural Science Foundation of China (62072265, 61572272, 71690231), and the MIIT High Quality Development Program 2020.

References

[1] Full version technical report. *https://sxsong.github.io/doc/ipm.pdf.*

[2] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[3] F. Bießmann, T. Rukat, P. Schmidt, P. Naidu, S. Schelter, A. Taptunov, D. Lange, and D. Salinas. Datawig: Missing value imputation for tables. *J. Mach. Learn. Res.*, 20:175:1–175:6, 2019.

[4] U. Brunner and K. Stockinger. Entity matching with transformer architectures - A step forward in data integration. In *EDBT*, pages 463–473, 2020.

[5] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *SIGMOD Conference*, pages 1335–1349, 2020.

[6] A. M. Dai and Q. V. Le. Semi-supervised sequence learning. In *NIPS*, pages 3079–3087, 2015.

[7] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, pages 4171–4186, 2019.

[8] C. Domeniconi and B. Yan. Nearest neighbor ensemble. In *ICPR (1)*, pages 228–231, 2004.

[9] M. Ebraheem, S. Thirumuruganathan, S. R. Joty, M. Ouzzani, and N. Tang. Distributed representations of tuples for entity resolution. *Proc. VLDB Endow.*, 11(11):1454–1467, 2018.

[10] L. Gondara and K. Wang. MIDA: multiple imputation using denoising autoencoders. In *PAKDD (3)*, volume 10939, pages 260–272, 2018.

[11] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. In *ACL (1)*, pages 328–339, 2018.

[12] T. Kenter, A. Borisov, and M. de Rijke. Siamese CBOW: optimizing word embeddings for sentence representations. In *ACL (1)*, 2016.

[13] B. Li, A. Drozd, Y. Guo, T. Liu, S. Matsuoka, and X. Du. Scaling word2vec on big corpus. *Data Sci. Eng.*, 4(2):157–175, 2019.

[14] D. Li, J. S. Deogun, W. Spaulding, and B. Shuart. Towards missing data imputation: A study of fuzzy k-means clustering method. In *Rough Sets and Current Trends in Computing*, volume 3066, pages 573–579. Springer, 2004.

[15] Y. Li, J. Li, Y. Suhara, A. Doan, and W. Tan. Deep entity matching with pre-trained language models. *CoRR*, abs/2004.00584, 2020.

[16] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[17] M. Mahdavi and Z. Abedjan. Baran: Effective error correction via a unified context representation and transfer learning. *Proc. VLDB Endow.*, 13(11):1948–1961, 2020.

[18] P. Mattei and J. Frellsen. MIWAE: deep generative modelling and imputation of incomplete data sets. In *ICML*, volume 97, pages 4413–4423, 2019.

[19] C. Mayfield, J. Neville, and S. Prabhakar. ERACER: a database approach for statistical inference and data cleaning. In *SIGMOD Conference*, pages 75–86, 2010.

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *ICLR (Workshop Poster)*, 2013.

[21] A. Nazábal, P. M. Olmos, Z. Ghahramani, and I. Valera. Handling incomplete heterogeneous data using vaes. *Pattern Recognit.*, 107:107501, 2020.

[22] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

[23] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang. Pre-trained models for natural language processing: A survey. *CoRR*, abs/2003.08271, 2020.

[24] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding with unsupervised learning. *Technical report, OpenAI*, 2018.

[25] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[26] P. Ramachandran, P. J. Liu, and Q. V. Le. Unsupervised pretraining for sequence to sequence learning. In *EMNLP*, pages 383–391, 2017.

[27] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *Proc. VLDB Endow.*, 10(11):1190–1201, 2017.

[28] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[29] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *ACL (1)*, 2016.

[30] M. D. Smucker, J. Allan, and B. Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 623–632, 2007.

[31] S. Song and Y. Sun. Imputing various incomplete attributes via distance likelihood maximization. In *KDD*, pages 535–545, 2020.

[32] S. Song, A. Zhang, L. Chen, and J. Wang. Enriching data imputation with extensive similarity neighbors. *Proc. VLDB Endow.*, 8(11):1286–1297, 2015.

[33] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[34] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. LINE: large-scale information network embedding. In *WWW*, pages 1067–1077, 2015.

[35] T. H. Trinh and Q. V. Le. A simple method for commonsense reasoning. *CoRR*, abs/1806.02847, 2018.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

[37] M. Wei, Y. He, and Q. Zhang. Robust layout-aware IE for visually rich documents with pre-trained language models. In *SIGIR*, pages 2367–2376, 2020.

[38] R. Wu, A. Zhang, I. F. Ilyas, and T. Rekatsinas. Attention-based learning for missing data imputation in holoclean. In *MLSys*, 2020.

[39] S. Wu, X. Feng, Y. Han, and Q. Wang. Missing categorical data imputation approach based on similarity. In *SMC*, pages 2827–2832, 2012.

[40] 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.

[41] D. Zhang, Z. Yuan, Y. Liu, Z. Fu, F. Zhuang, P. Wang, H. Chen, and H. Xiong. E-BERT: A phrase and product knowledge enhanced language model for e-commerce. *CoRR*, abs/2009.02835, 2020.

[42] S. Zhang, J. Zhang, X. Zhu, Y. Qin, and C. Zhang. Missing value imputation based on data clustering. *Trans. Comput. Sci.*, 1:128–138, 2008.

[43] Y. Zhu, R. Kiros, R. S. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *ICCV*, pages 19–27, 2015.