A Region-Splitting Imputation method for different types of missing data

1. Introduction

2. Related work

3. Preliminaries

- In region-splitting imputation(RESI) model, imputation and partition are the most critical links. Therefore, the selection of base imputing technology and weight calculation in partitioning algorithm are both crucial.

- The K-Nearest Neighbor Imputation(KNNI) method and the Entropy Weight Method are selected as the base imputer and the way to calculate weight respectively.

3.1. KNNI

- Is a commonly used non-parametric imputation method, whose idea is as follow:
  1. Given a test sample of missing value
  2. Select the K nearest samples in the observed set (complete samples)
  3. Use some distance measurement (generally Euclidean distance)
  4. Predict according to the information of the k neighbors.

- For categorical data, it uses the voting method to choose tag.
- For numerical data, the average method can be adopted, that is, the average value or weighted average value of the k nearest neighbors.

3.1. KNNI

- with other prediction models, it will cost a large amount of time to train the model in each iteration.

- Different from other training methods, KNN, as a famous representative of lazy learning has almost no explicit training process.

- In the training stage, it only saves the observed samples and processes them after receiving the test samples.
- The training time cost of this learning technology is almost zero.

- which effectively solves the computational inefficiency that may occur in the learning method

3.2. Entropy weight method (EWM)

- Determine the weight of each attribute.
- Its main idea is to decide the objective weight according to the magnitude of attribute variability.

- Generally, low information entropy indicates that the attribute has a large variability and it can provide more information for the following data analysis, which means it has greater impact on the data mining results and its weight naturally increases.

3.2. Entropy weight method (EWM)

- Step 1: Normalizing data. Given s attributes $A1$, $A2$, … , Ai , … , As , where Ai = {$ai1$, $ai2$, … , aij, … , ain}, we denote the normalized value of attribute Ai as Yi = {$yi1$, $yi2$, …, yin} and each $yij \in Yi$ is formulized as:
    - $yij = (aij - min(Ai\ )) \ / \ (max(Ai\ ) - min(Ai\ )\ )$

- Step 2: Calculating the entropy of each attribute. Suppose Yi is the normalized set of the attribute Ai , the possibility of each value $aij \in Ai$ can be computed as $pij = Yij \sum n\ i{=}1\ Yij$. Specially, when pij = 0, lim pij→0 $pijlnpij$ = 0. Thus, the entropy of the attribute Ai can be calculated as:
    - $Ei = -ln(n)\ {-1} \sum n\ i{=}1\ pij\ .\ ln\ pij$

- Step 3: Determining the weight of each attribute. After computing the entropy of each attribute with Formula (2), next is to compute the weight of each attribute with :
    - $wi = 1 - Ei\ k - \sum Ei\ (i = 1, 2, \dots , k)$

3.3. Problem definition

- Given a dataset contains lots of tuples, each of which is composed of several attributes. The attribute is in either numerical or categorical type.
- Suppose there are several various types of values missed in part of tuples.

- our goal is to establish a model which can effectively complete the tuples with numerical only, categorical only or mixed-type of both.

we formally define the problem as follows:

- Definition 1 (Complete Tuple and Incomplete Tuple).
  - The definition of CT and ICT implies CTs have no missing items and ICTs have at least one missing item.
- Definition 2 (Missing Rate).
  - represents the percentage of its missing items in its total items. $MRate_D = \frac{N_{miss}}{n * s}$
  - where N_{miss} is the number of missing items in dataset.
- Definition 3 (Complete Ratio).
  - refers to the proportion of complete tuples in dataset. $CRatio_D = \frac{CT_D}{CT_D + ICT_D}$
  - in which, CT_D and ICT_D are the set of complete and incomplete tuple of D, respectively.

we formally define the problem as follows:

- Definition 4 (Missing Type).
  - can be divided into two subsets, that is, $M_D = M_{num} \cup M_{cat}$ where M_{num} and M_{cat} stand for numerical and categorical missing values respectively, and obviously $M_{num} \cap M_{cat} = \phi$.
- missing type of D can be classified as:
  1. Categorical-missing dataset
  2. Numerical-missing
  3. Mixed-type missing dataset

- In reality, it is more common for the above three types of missing values to exist simultaneously.

THE GOAL OF THIS ARTICLE IS TO DESIGN AND IMPLEMENT A SOLUTION THAT CAN EFFECTIVELY IMPUTE THESE THREE DIFFERENT TYPES OF MISSING VALUES AT THE SAME TIME.

4. Our model

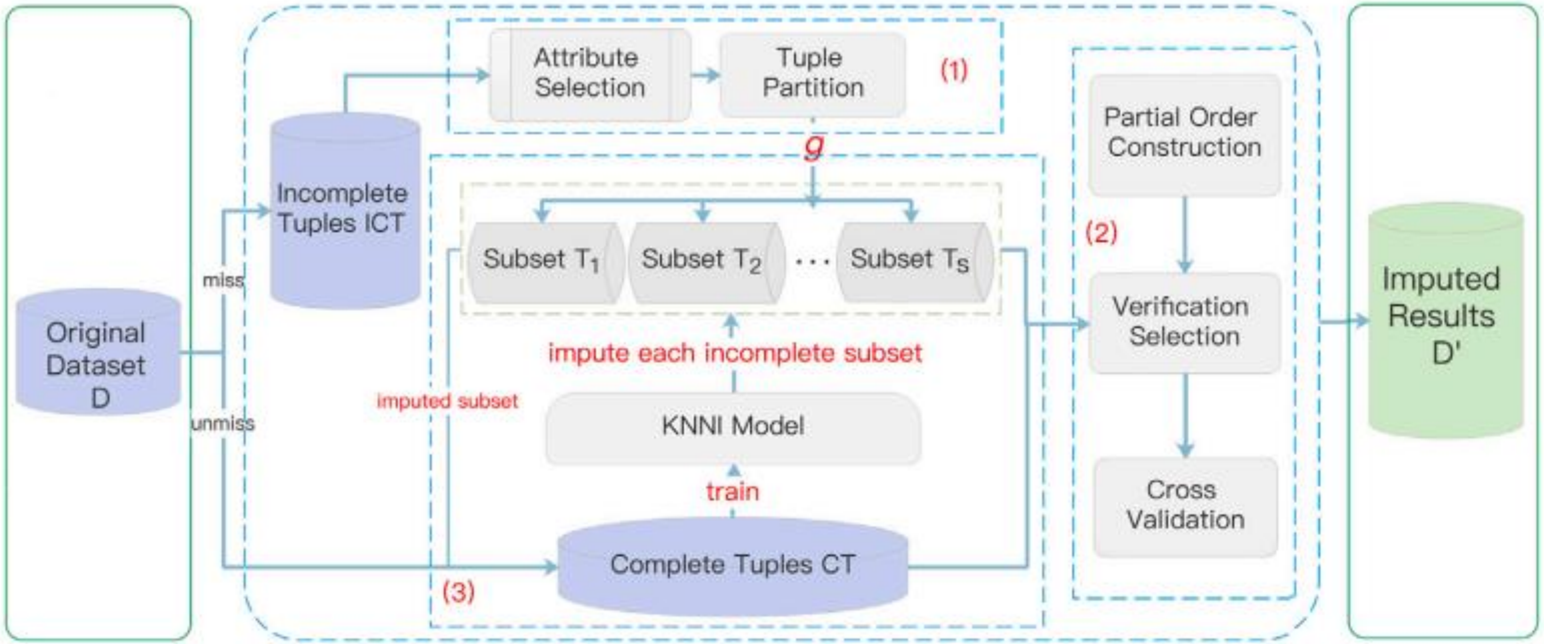

Fig. 1. Overview of RESI framework.

4.1. Framework of RESI

1. Divide the input dataset into two parts: complete tuples & incomplete tuples.

- CT is used as the training set for KNNI
- ICT are partitioned into several subsets, named incomplete subsets according to their missing degree. Each incomplete subset is respectively taken as a test set and filled with the previously trained KNNI model.

4.1. Framework of RESI

- RESI accomplishes the imputation in three steps:
  1. tuple partition
  2. incomplete subsets imputation
  3. cross correction.

- The final filling results are obtained by averaging the predicted values of (2) and (3).

Algorithm 1: Algorithm of Subset Imputation

Input: D: a dataset of relation schema having missing value to impute
m: number of incomplete subsets
Output: D': dataset with filled values

Begin:
CT_0=ExtractCompleteTuples(D)
ICT=ExtractIncompleteTuples(D)
GenerateTuplePartitions(ICT, method=EWM, output= $\mathcal{T}\{T_1, T_2, \cdots, T_m\}$)
for i in 1:m **do**
    T_i'=KNNImputation(train_set=CT_{i-1}, test_set=T_i)
    CT_i=merge(CT_{i-1}, T_i')
Do cross validation:
for i in 1:m-1 **do**
    T_i''=KNNImputation(train_set=CT_m, test_set=T_i)
T_m''=KNNImputation(train_set=CT_0, test_set=T_m)
for i in 1:m **do**
    CT_i=merge(CT_{i-1}, mean(T_i', T_i''))
$D' = CT_m$
return D'

4.2. Tuple partition

- It is not reasonable to deal with all incomplete tuples at the same time during the imputing process.

- Practice shows that different attributes have different effects on subsequent analysis

- In a dataset instance, different missing items of tuples will also have different impact on subsequent modeling.
- So we define an indicator called tuple integrity rate.

Integrity rate

- Definition 5 (Tuple Integrity Rate).

- Example of computing tuple integrity rate
  - In current example, the importance of birthplace varies from that of workplace in the modeling of wages. Assuming that with EWM, we compute the weight of birthplace is 0.10 and that of workplace is 0.11, and the birthplace is a missing item in tuple ta , while the workplace is a missing item in tuple tb . Both ta and tb have no other missing items. Then, the tuple integrity rate of ta and tb are 0.90 and 0.89, respectively.

Algorithm 2: Algorithm of Generating Tuple Partitions.

Input: CT: subset with complete tuples; ICT: subset with incomplete tuples; m: number of incomplete subsets; s: number of attributes

Output: $\mathcal{T}$: a queue of subsets

Begin:
W=ComputeAttributeWeights (train_set=CT, method=EWM, output=$\{w_1, w_2, \cdots, w_s\}$)
Do Intergrity Computation:
for t_i in ICT **do**
    $t_i = \{a_1, a_2, \cdots, a_s\}$
    $r_i = 1$
    for j in 1: s **do**
        if a_1=0 **then**
            $r_i = r_i - w_j$
$\mathcal{P}$=SortIncompleteTuples(object=ICT, by=r, order=descending)
$\mathcal{T}$=GenerateTuplePartition($\mathcal{P}$,m)
return $\mathcal{T}$

4.3. Incomplete subsets imputation

4.4. Cross correction

4.5. Complexity of algorithm

1. Weight calculation → O(s × t)
   - All date is traversed and normalized.
2. Tuple Partition → O(s × t)
   - Each tuple in the dataset is traversed to determine if it is an incomplete tuple.
   - Calculating the integrity rate
3. Missing value imputation → O(n × t)
   - It is necessary to traverse and fill all missing items in turn, and cross-verify the correction strategy for each tuple.

- Hence, the overall time complexity is $T = O(s \times t) + O(s \times t) + O(n \times t)$, and when $n \gg s$, the complexity approximates $O(n \times t)$.

5. Experiments and analysis

6. Conclusion

Thanks