Privacy-Preserving Data Imputation*

Geetha Jagannathan
Stevens Institute of Technology
Hoboken, NJ, 07030, USA
gjaganna@cs.stevens.edu

Rebecca N. Wright
Stevens Institute of Technology
Hoboken, NJ, 07030, USA
rwright@cs.stevens.edu

Abstract

In this paper, we investigate privacy-preserving data imputation on distributed databases. We present a privacy-preserving protocol for filling in missing values using a lazy decision tree imputation algorithm for data that is horizontally partitioned between two parties. The participants of the protocol learn only the imputed values; the computed decision tree is not learned by either party.

1. Introduction

Real-world databases are frequently "dirty"—the data is noisy, inconsistent, and has missing values. If this unprocessed raw data is used as input for data mining processes, the extracted knowledge is likely to be of poor quality as well. *Data cleaning* is a preprocessing step in data mining that smooths noise in the data, identifies and eliminates inconsistencies, and replaces missing values (also called "data imputation"). While much of data mining, including data cleaning, occurs on data within an organization, it is quite common to use data from multiple sources in order to yield more precise or useful knowledge. However, privacy concerns and privacy regulations often prevent the sharing of data between multiple parties. Since Yao's general purpose secure circuit-evaluation protocol [16] is impractical for large datasets, secure special-purpose protocols have been developed for specific data mining problems, initiated by [1, 10].

Data imputation is the process of replacing missing values with estimated values. Imputation techniques range from simple ideas such as using the mean or mode of the attribute for a missing value [8] to more sophisticated ones [2, 3, 9]. Using the mean or mode is generally considered a poor choice [11], as it distorts other statistical properties of the data and does not take dependencies between attributes into account. Hot-deck imputation [5] fills in a missing value using values from other rows of the database that are similar to the row with the missing value.

Classification is generally considered the best method for imputing missing data [4]. The most commonly used methods are regression-based imputation [2] and decision-tree-based imputation [9, 4]. Regression assumes a specific relationship between attributes that may not hold for all data sets. Decision-tree imputation uses a decision-tree based learning algorithm such as ID3 [15] to build a decision-tree classifier using the rows with no missing values, with the attribute that has the missing value as the class attribute. The tree is evaluated on the row with the missing value to predict the missing value. For efficiency purposes, the decision tree construction can be *lazy* [6], in that only the needed path of the tree is constructed. Lindell and Pinkas [10] provide a privacy-preserving algorithm for computing the ID3 tree for horizontally partitioned databases. Our privacy-preserving data imputation solution also uses ID3 trees, but it differs from their algorithm in that we only construct the path needed and we are able to use the path for classification without either party learning the constructed path.

In this paper, we provide a privacy-preserving solution to the data imputation problem. To our knowledge, this is the first paper addressing privacy-preserving methods of preprocessing data. We present a privacy-preserving data imputation protocol for databases that are horizontally partitioned between two parties. Our protocol uses a lazy decision tree algorithm based on ID3 trees. It allows either party to compute missing values without requiring the parties to share any information about their data and without revealing the decision tree or the traversed path to either party. Our protocol reveals the number of nodes traversed by the protocol in the undisclosed decision tree used for imputation. With some increased computation and communication cost, this protocol can be modified so that it does not leak any information beyond the computed imputed value. (Due to space limitations, we do not provide this extension here.)

*This work was supported by the National Science Foundation under Grant No. CCR-0331584.

In *Proceedings of the ICDM International Workshop on Privacy Aspects of Data Mining*, December 18, 2006.

Algorithm Lazy Decision Tree

Input: A database D of labeled instances (with attributes $R = \{A_1, \ldots, A_k\}$) and an unlabeled instance I to be classified.

Output: A label for instance I.

1. If R is empty, return the majority label of transactions in D.
2. If D is pure, return the label c.
3. Otherwise,

(a) for i = 1 to k
Entropy(A_i) = Calc_Split_Entropy(D, A_i)
(b) $A_{\min}$ = Attribute with least entropy.
(c) $D \leftarrow \{X \in D \mid X[A_{\min}] = I[A_{\min}]\}$
(d) $R \leftarrow R - \{A_{\min}\}$

4. Go to Step 2

Subroutine Calc_Split_Entropy

Input: A database D of labeled instances and an attribute A

Output: Entropy after splitting D on attribute A.

1. Let A take values $a_1, \ldots, a_m$.
2. Entropy(D, A) =
$$-\sum_{j=1}^{m} \frac{|D(a_j)|}{|D|} \left(\frac{p_{j0}}{|D(a_j)|} \log \frac{p_{j0}}{|D(a_j)|} + \frac{p_{j1}}{|D(a_j)|} \log \frac{p_{j1}}{|D(a_j)|} \right)$$

where
$D(a_j)$ = instances of D in which A has value a_j,
p_{jc} = # instances of $D(a_j)$ in which class label is $c \in \{0, 1\}$.

3. return Entropy(D, A)

Figure 1. The Lazy Decision Tree Algorithm

2. Preliminaries

LAZY DECISION TREE ALGORITHM. Our lazy decision tree algorithm is a simplification of ID3 that lends itself to an efficient privacy-preserving distributed solution. Our algorithm (Figure 1) fits the definition of Friedman et al. [6] for a generic lazy decision tree algorithm, though it differs significantly from the specific LazyDT algorithm presented in that paper. LazyDT is more complex, slower, and not easily amenable for conversion to a privacy-preserving protocol. The experiments in [6] indicate that LazyDT, on average, has slightly better accuracy than ID3 (84% for LazyDT vs. 80% for ID3). As in any lazy learning algorithm, our algorithm does not create an explicit model from the training data. Instead, the test instance to be classified is used to directly trace the path that would have been taken if an ID3 tree had been built from the training data. A database D is said to be *pure* if all the transactions in D have the same class label. In what follows, we assume that the class attribute is binary, though our solution can be modified to handle any finite class attribute.

OUR MODEL. In our setting, two parties, Alice and Bob, own databases $D_A = (d_1, \ldots, d_\ell)$ and $D_B = (d_{\ell+1}, \ldots, d_n)$, respectively, defined over a common set of attributes $\{A_1, \ldots, A_k\} \cup M$. We use the notation $\alpha \in$ Alice to indicate that Alice holds the value α and we use a similar notation for Bob. Suppose $I \in$ Bob (not included in $\{d_1, \ldots, d_n\}$) has a missing value for the attribute M. Bob wishes to compute the missing value $I(M)$ using $D = D_A \cup D_B$ via a data imputation algorithm agreed to by both Alice and Bob. No other information is to be revealed to either party. In this paper, we assume that attribute M takes the values 0 and 1; it is easy to extend the protocol to the case where M takes more than two values.

Privacy definitions are given in relation to an ideal model, in which there is a trusted third party to whom Alice and Bob send their data. The third party uses the imputation algorithm chosen by Alice and Bob to compute a missing value and sends the computed value to Bob. In a private protocol, Alice and Bob compute the missing value by solely communicating with each other instead of using the trusted third party; in doing so, they should not learn anything that they would not learn in the ideal model. In this paper, we assume that both Alice and Bob are *semi-honest*. That is, both parties faithfully follow their specified protocols, but we assume they record intermediate messages in an attempt to infer information about the other party's data.

In the discussions of our protocols and their analysis, n denotes the size of the database, k denotes the number of attributes, m denotes the maximum number of values any attribute can take, and c denotes the maximum number of bits required to represent any encryption.

CRYPTOGRAPHIC PRIMITIVES.

We say that *Alice and Bob have random shares of a value x drawn from a field F of size N* to mean that the value x is divided into two pieces $a, b \in F$ such that Alice knows a, Bob knows b, and x can be recovered from a and b. We use additive sharings and XOR sharings. We choose N to be a large prime and the field $F = \mathbb{Z}_N$. In additive sharing, a value x is shared as $(a+b) \bmod N \equiv x$ for random a and b. In XOR sharing, a bit x is shared as $x = a \oplus b$ for random a and b. Except where otherwise specified, all computations throughout the paper take place in F.

We also make use of the private indirect index protocol [12]. In this protocol, Bob has a vector of values $X = (x_1, \ldots, x_n)$. Alice and Bob wish to compute random shares of x_i, where input i is available as a random XOR sharing between Alice and Bob (that is, $i = i_1 \oplus i_2$, where $i_1 \in$ Alice and $i_2 \in$ Bob). Although the protocol in [12] outputs an XOR-sharing of x_i, it can be easily modified to provide an additive sharing of the output. This pro-

tocol makes use of one invocation of 1-out-of-n oblivious transfer (OT_1^n) [13]. OT_1^n is a two party protocol where the sender has n inputs $\{x_1, \ldots, x_n\}$ and the receiver has an input $j \in \{1, \ldots, n\}$. At the end of the protocol, the receiver learns x_j and no other information; the sender learns nothing. We also use semantically secure additively homomorphic encryption, (e.g., [14]), private scalar product computation [7], and Yao's two-party secure circuit-evaluation protocol [16] (used only on small circuits).

3. Privacy-Preserving Imputation Based on Lazy Decision Trees

Our privacy-preserving data imputation protocol is based on the lazy decision tree algorithm described in Section 2. As described, our protocol reveals the number of nodes traversed by the protocol in the undisclosed decision tree used for imputation. This leak can be removed at a slightly increased cost of communication and computation.

3.1. Our Basic Protocol

We rephrase the basic steps of the lazy decision tree algorithm from Section 2 so that it functions more clearly as a data imputation algorithm. The lazy decision tree algorithm first computes as the root of the tree the attribute with the highest information gain. This is followed by the repeated execution of the following steps until the remaining instances are pure or all attributes have been exhausted:

1. Extract the subset of the instances that match I on the chosen attribute.

2. Choose an attribute of high information gain for the next iteration.

The algorithm outputs as the missing value the majority label on the remaining instances.

Our privacy-preserving protocol follows these same steps. However, the privacy requirements prohibit the protocol from revealing any information including the attributes that have been chosen along the way and the subset of the instances that match I on the chosen attributes. The protocol handles the first condition by storing the index s of the chosen attribute A_s in any iteration as a random sharing between the two parties. That is, Alice learns s_A and Bob learns s_B such that $s_A \oplus s_B = s$. To satisfy the second condition, the protocol uses a randomly shared bit-vector representation of any $D' \subseteq D$. That is, Alice and Bob have, respectively, bit-vectors $(p_1, \ldots, p_n)$ and $(q_1, \ldots, q_n)$ such that for any $d_i \in D$:

$$p_i \oplus q_i = \begin{cases} 1 & \text{if } d_i \in D' \\ 0 & \text{otherwise} \end{cases}$$

The entropy computed for each attribute in each iteration is also held as random shares between the two parties.

We now describe in more detail the protocol in Figure 2. At the beginning of the protocol, Alice and Bob jointly check if the database D is pure. If D is pure, Bob outputs the majority label. This is done without revealing either their data or one-sided information about purity to each other using Step 2 of Protocol 1 in [10].

To compute the root, Alice and Bob compute random shares of the entropy for each attribute A_i. Using the protocol that computes a random sharing of $x \log x$ [10] given a sharing of x, Alice and Bob jointly compute a random sharing of $\mathsf{Entropy}(D, A_i)$ as (Ent_i^A) and (Ent_i^B) respectively such that $\mathsf{Ent}_i^A + \mathsf{Ent}_i^B \equiv \mathsf{Entropy}(D, A_i) \bmod N$. The index of the attributes that yields the least entropy is computed as random shares ($\min_A \in$ Alice and $\min_B \in$ Bob such that $\min = \min_A \oplus \min_B$) between Alice and Bob using Yao's protocol. Note that either $\mathsf{Ent}_i^A + \mathsf{Ent}_i^B = \mathsf{Entropy}(D, A_i)$ or $\mathsf{Ent}_i^A + \mathsf{Ent}_i^B = \mathsf{Entropy}(D, A_i) + N$. The circuit first computes $\mathsf{Ent}_i^A + \mathsf{Ent}_i^B$; if the result is greater than $N - 1$, the circuit subtracts N. It selects the attribute with the lowest entropy.

We write D_j to denote the subset of D that has "filtered" through to level j of the lazy decision tree. To compute the attribute at level j, Alice and Bob should split the database D_{j-1} on the attribute A_s chosen at level $j-1$. This involves finding $I[A_s]$. Here the instance I is known to Bob, but s is randomly shared between Alice and Bob. Alice and Bob compute a random sharing of $I[A_s]$ using the private indirect indexing protocol [12]. Since D_j should be unknown to either party, we store the set in the form of two bits $p_i \in$ Alice and $q_i \in$ Bob per record ($1 \leq i \leq n$) such that

$$p_i \oplus q_i = \begin{cases} 1 & \text{if } d_i[A_s] = I[A_s] \text{ and } d_i \in D_{j-1} \\ 0 & \text{otherwise} \end{cases}$$

(See Section 3.3.) Note that the information about the inclusion of d_i in D_{j-1} is also shared between Alice and Bob. For the root level (which contains all of D), we set $p_i = 1$ and $q_i = 0$ for all i.

If D_j is pure, then Bob outputs this majority value. It is important to observe that since neither party knows which instances are in D_j, we cannot use the purity check in Step 2 of Protocol 1 in [10]. We provide an alternate purity checking protocol in Section 3.5. If D_j is not pure, both parties securely compute the random shares of the index of the attribute with least entropy. To simplify the protocol, the entropy is evaluated for all attributes at all levels of the decision tree. This does not impact the correctness of the protocol, as the information gain would be zero for an attribute that has already been chosen at a previous level. We present the privacy-preserving lazy decision tree imputation protocol in Figure 2.

Protocol Private Lazy Decision Tree Data Imputation

Input: A database $D = D_A \cup D_B$ of labeled instances (with attributes $\{A_1, \ldots, A_k\} \cup M$), where Alice owns $D_A = (d_1, \ldots, d_\ell)$ and Bob owns $D_B = (d_{\ell+1}, \ldots, d_n)$ and an instance I with a missing value $I(M)$.

Output: Bob outputs $I(M)$.

1. If D is pure, Bob outputs the majority label and exits.

2. Alice and Bob communicate with each other to compute the root attribute using the following steps:

(a) for i = 1 to k, Alice and Bob compute random shares of $\mathsf{Entropy}(D, A_i)$ as $\mathsf{Ent}_i^A + \mathsf{Ent}_i^B \equiv \mathsf{Entropy}(D, A_i) \bmod N$.
(b) Using Yao's protocol, Alice and Bob compute $\min_A$ and $\min_B$ such that $\min_A \oplus \min_B = \min$ where $\mathsf{Ent}_{\min}^A + \mathsf{Ent}_{\min}^B \equiv \text{minimum}\{\mathsf{Ent}_i^A + \mathsf{Ent}_i^B\}, 1 \le i \le k$.

3. for j = 1 to $k - 1$

(a) Alice and Bob jointly compute the set $D_j = \{d \in D_{j-1} \mid d[A_{\min}] = I[A_{\min}]\}$ (represented by bit-vectors P and Q) as follows:

- Alice and Bob compute random shares of $I[A_{\min}]$ as $\alpha \in$ Alice and $\beta \in$ Bob using the private indirect indexing protocol [12] with inputs $\min_A \in$ Alice and I, $\min_B \in$ Bob.
- Alice and Bob run the secure split computation protocol (see Section 3.3) on each of their records and α, β to obtain two bit vectors $P = (p_1, \ldots, p_n) \in$ Alice and $Q = (q_1, \ldots, q_n) \in$ Bob such that $p_i \oplus q_i = 1$ if $d_i[A_{\min}] = I[A_{\min}]$, and 0 otherwise.

(b) If D_j is pure, Bob outputs the majority label and exits (see Section 3.5). Otherwise:

- for i = 1 to k, $\mathsf{Ent}_A^i + \mathsf{Ent}_B^i = \mathsf{Secure\_Ent\_Comp}(D_j, P, Q, A_i)$
- Using Yao's protocol, Alice and Bob compute $\min_A$ and $\min_B$ such that $\min_A \oplus \min_B = \min$ where $\mathsf{Ent}_{\min}^A + \mathsf{Ent}_{\min}^B \equiv \text{minimum}\{\mathsf{Ent}_i^A + \mathsf{Ent}_i^B\}, 1 \le i \le k$.

4. Bob outputs the majority label as the missing value using the secure majority computation protocol (see Section 3.6).

Figure 2. Private Lazy Decision Tree Imputation Protocol

Protocol Secure_Ent_Comp

Input: A database D of labeled instances where Alice owns $D_A = (d_1, \ldots, d_\ell)$ and Bob owns $D_B = (d_{\ell+1}, \ldots, d_n)$, an attribute A known to both Alice and Bob, and $D' \subseteq D$ represented by bit vectors $P = (p_1, \ldots, p_n)$ and $Q = (q_1, \ldots, q_n)$ belonging to Alice and Bob, respectively, such that for $1 \le i \le n$, $p_i \oplus q_i = 1$ if $d_i \in D'$ and 0 otherwise.

Output: Random shares of entropy after splitting D' on attribute A.

1. Let A take values $a_1, \ldots, a_m$.
2. for j = 1 to m
   (a) Compute $D(a_j) \subseteq D'$ in which A has value a_j. $D(a_j)$ is represented as two bit vectors $R \in$ Alice and $S \in$ Bob such that, for $1 \le i \le n$,
   $$R_i \oplus S_i = \begin{cases} 1 & \text{if } p_i \oplus q_i = 1 \text{ and } d_i[A] = a_j \\ 0 & \text{otherwise} \end{cases}$$
   To compute R and S, for each i, Alice and Bob use Yao's protocol where Alice inputs p_i and Bob inputs q_i. If $d_i \in$ Alice, then she inputs $d_i[A]$; otherwise, Bob inputs $d_i[A]$. Alice and Bob output R_i and S_i, respectively.
   (b) Alice and Bob engage in the secure Total_Record_Split protocol and compute the random shares of $|D(a_j)|$.
   (c) Let p_{j0} = # instances of $D(a_j)$ in which label is 0. Alice and Bob engage in the protocol described in Section 3.4.2 and output random shares of p_{j0}.
   (d) Let p_{j1} = # instances of $D(a_j)$ in which label is 1. Alice and Bob engage in the protocol described in Section 3.4.2 and output random shares of p_{j1}.
   (e) They use the secure $x \log x$ protocol [10] to compute random shares of $|D(a_j)| \log |D(a_j)|$, $p_{jk} \log p_{jk}$ for $k = 0, 1$.
   (f) They compute random shares of $\mathsf{sum}_j = |D(a_j)| \log |D(a_j)| - \sum_{k=0}^{1} (p_{jk} \log p_{jk})$.
3. Alice and Bob compute random shares of entropy for the attribute A by adding the shares obtained in Step 2f.

Figure 3. Secure Split Entropy

3.2. Secure Protocol to Compute Split Entropy

This protocol takes a subset D' of the database D, horizontally partitioned between Alice and Bob, and an attribute A known to both Alice and Bob. D' is represented by two bit vectors P and Q known to Alice and Bob, respectively, such that for $1 \leq i \leq n$,

$$p_i \oplus q_i = \begin{cases} 1 & \text{if } d_i \in D' \\ 0 & \text{otherwise} \end{cases}$$

The protocol outputs random shares of the entropy after splitting D' on the attribute A. We present this protocol in Figure 3.

3.3. Secure Split Protocol

The secure split protocol is used to find whether a given record is in a subset $D' \subseteq D$ and attribute A_i takes the value α in that record. The result of this test (0 or 1) is randomly shared between Alice and Bob. Here, inputs α and i are shared between Alice and Bob as $\alpha \equiv a + b \bmod N$ and $i = i_1 \oplus i_2$, where $a, i_1 \in$ Alice and $b, i_2 \in$ Bob. The inclusion of the given record in D' is represented by two bits p and q, known to Alice and Bob respectively, such that $p \oplus q = 1$ if the record is in D' and 0 otherwise.

Assuming that the record belongs to Alice, Alice's inputs to the protocol are the record $(v_1, \ldots, v_k)$, a, i_1, and p. Bob's inputs are b, i_2 and q. (The case where Bob owns the record is similar.) At the end of this protocol, Alice and Bob receive bits b_1 and b_2, respectively, such that

$$b_1 \oplus b_2 = \begin{cases} 1 & \text{if } v_i = \alpha \text{ and } p \oplus q = 1 \\ 0 & \text{otherwise} \end{cases}$$

This protocol has two stages. In the first stage, Alice and Bob use the private indirect indexing protocol [12] and output v and w, respectively, where $v + w \equiv v_i \bmod N$. In the second stage, Alice has inputs v, a and p, and Bob has inputs w, b and q. They use Yao's protocol and output two bits $b_1 \in$ Alice and $b_2 \in$ Bob such that

$$b_1 \oplus b_2 = \begin{cases} 1 & \text{if } v + w \equiv a + b \bmod N \text{and } p \oplus q = 1 \\ 0 & \text{otherwise} \end{cases}$$

3.4. Secure Information Gain Protocol

In this subsection, we present protocols that are used in computing the information gain for each attribute.

3.4.1 Secure computation of the total number of records in the split

This protocol takes as input a subset $D' \subseteq D$ and computes shares of $|D'|$. D' is represented by bit vectors $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ known to Alice and Bob, respectively, where

$$x_i \oplus y_i = \begin{cases} 1 & \text{if instance } d_i \in D' \\ 0 & \text{otherwise} \end{cases}$$

At the end of the protocol, Alice and Bob output random shares of $|D'|$. We present this protocol in Figure 4.

Protocol Total_Record_Split

Input: Alice has a bit vector $X = (x_1, \ldots, x_n)$, Bob has a bit vector $Y = (y_1, \ldots, y_n)$,

Output: Alice and Bob output α and β, respectively, such that $\alpha + \beta \equiv |D'| \bmod N$, where $|D'|$ is the total number of records for which $x_i \oplus y_i = 1$.

1. Alice computes $\sum_{i=1}^{n} x_i = \gamma$.
2. Bob computes $\sum_{i=1}^{n} y_i = \delta$.
3. Alice and Bob securely compute the shares of the scalar product of the vectors X and Y as $\mu \in$ Alice and $\lambda \in$ Bob.
4. Alice outputs $\alpha = \gamma - 2\mu$, Bob outputs $\beta = \delta - 2\lambda$.

Figure 4. Secure Computation of $|D'|$

3.4.2 Secure computation of the total number of records in the split with a given class value

This protocol takes as input a subset $D' \subseteq D$ and computes shares of the total number of records in the set D'_c. Here, D'_c denotes the set of records in D' in which the attribute M takes the value c ($c \in \{0, 1\}$). D' is represented by the bit vectors $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$ known to Alice and Bob, respectively, where

$$x_i \oplus y_i = \begin{cases} 1 & \text{if instance } d_i \in D' \\ 0 & \text{otherwise} \end{cases}$$

Alice computes a vector $W = (w_1, \ldots, w_\ell)$, such that, for $1 \leq i \leq \ell$,

$$w_i = \begin{cases} 1 & \text{if } d_i[M] = c \\ 0 & \text{otherwise.} \end{cases}$$

Define $|D'_{Ac}|$ as the total number of records owned by Alice in which $x_i \oplus y_i = 1$ (the record is in D') and $w_i = 1$ (attribute M takes the given value). That is, $|D'_{Ac}|$ is the number of records in which $(\overline{x_i} w_i) y_i \vee (x_i w_i) \overline{y_i} = 1$. Alice and Bob run the scalar product protocol [7] once with inputs $(\overline{x_1} w_1, \ldots, \overline{x_\ell} w_\ell)$ and $(y_1, \ldots, y_\ell)$, respectively, to obtain shares α_1 and β_1, respectively. They run the scalar product protocol a second time with inputs $(x_1 w_1, \ldots, x_\ell w_\ell)$ and $(\overline{y_1}, \ldots, \overline{y_\ell})$, respectively, to obtain shares α_2 and β_2 respectively. Alice's share of $|D'_{Ac}|$ is $\alpha_1 + \alpha_2$ and Bob's share is $\beta_1 + \beta_2$. Similarly, Alice and Bob compute shares of $|D'_{Bc}|$. Alice and Bob add their respective shares to obtain shares of $|D'_c|$.

3.5. Secure Protocol to Check If D is Pure

This protocol takes as input a subset D' of a database D and outputs c if all the transactions in D' have the same label c or $\perp$ otherwise. D' is represented by two bit vectors $P = (p_1, \ldots, p_n) \in$ Alice and $Q = (q_1, \ldots, q_n) \in$ Bob such that

$$p_i \oplus q_i = \begin{cases} 1 & \text{if } d_i \in D' \\ 0 & \text{otherwise} \end{cases}$$

Alice and Bob compute random shares of $|D'_c|$, for $c = 0, 1$ using the protocol described in Section 3.4.2. Let α_{Ac} and α_{Bc} denote Alice's and Bob's shares for $c = 0, 1$, respectively. Alice and Bob use Yao's protocol to check if $\alpha_{A0} + \alpha_{B0} \equiv 0 \bmod N$. If so, Bob outputs 1. If $\alpha_{A1} + \alpha_{B1} \equiv 0 \bmod N$, then Bob outputs 0. Otherwise, he outputs $\perp$.

3.6. Secure Protocol for Majority Computation

This protocol is similar to the protocol that checks if D is pure (Section 3.5). In the first stage, Alice and Bob compute random shares of $|D_0| = \alpha_{A0} + \alpha_{B0} \bmod N$ and $|D_1| = \alpha_{A1} + \alpha_{B1} \bmod N$, where α_{A0}, $\alpha_{A1} \in$ Alice and $\alpha_{B0}, \alpha_{B1} \in$ Bob, using the protocol described in Section 3.4.2. In the second stage, Alice and Bob invoke the secure circuit evaluation of Yao's protocol with Alice's input as α_{A0}, α_{A1} and Bob's input as α_{B0}, α_{B1}. Bob outputs the majority label as 1 if $|D_0| < |D_1|$ or 0 otherwise.

3.7. Performance Analysis

The communication complexity is dominated by $O(cnk^2m)$ + $O(mk^2S \log n)$, where S is the length of the key of the pseudorandom function used in the $x \log x$ protocol [10]. The total computation complexity at the end of the protocol is given by $O(k^2m \log n)\ OT_1^2$ + $O(k)\ OT_1^k$ + $O(k^2m)$ executions of the scalar product protocol (which involves $O(k^2nm)$ encryptions and $O(k^2m)$ decryptions for Alice and $O(k^2nm)$ exponentiations and $O(k^2m)$ encryptions for Bob).

In our privacy-preserving imputation protocol, the attribute and the split of the database at each level of the path in the decision tree are available only as random shares to Alice and Bob. In addition, all the intermediate outputs of the subprotocols are held only as random shares by Alice and Bob. Composing the subprotocols into the entire protocol, Bob learns the desired missing value. However, in addition, both Alice and Bob learn the number of attributes (though not which attributes, nor the values they take) in the path in the decision tree, which would not be learned if a trusted third party were used in the computation. Our protocol can be easily modified to prevent Alice and Bob from learning the number of attributes in the path.

References

[1] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *ACM SIGMOD*, pages 439–450, May 2000.

[2] J. Beaumont. On regression imputation in the presence of nonignorable nonresponse. In *The Survey Research Methods Section*, pages 580–585. ASA, 2000.

[3] L. Coppola, M. D. Zio, O. Luzi, A. Ponti, and M. Scanu. Bayesian networks for imputation in official statistics: A case study. In *DataClean Conference*, pages 30–31, 2000.

[4] A. Farhangfar, L. Kurgan, and W. Pedrycz. Experimental analysis of methods for handling missing values in databases. In *Intelligent Computing: Th. and Appl. II*, 2004.

[5] B. L. Ford. *Incomplete data in sample surveys*, chapter An overview of hot-deck procedures. Academic Press, 1983.

[6] J. H. Friedman, R. Kohavi, and Y. Yun. Lazy decision trees. In *13th Art. Intell. and the 8th Innovative Applications of Art. Intell.*, pages 717–724, 1996.

[7] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikainen. On secure scalar product computation for privacy-preserving data mining. In *7th ICISC*, 2004.

[8] M. Hu, S. Salvucci, and M. Cohen. Evaluation of some popular imputation algorithms. In *The Survey Research Methods Section of the ASA*, pages 308–313, 1998.

[9] K. Lakshminarayan, S. A. Harp, and T. Samad. Imputation of missing data in industrial databases. *Applied Intelligence*, 11(3):259–275, 1999.

[10] Y. Lindell and B. Pinkas. Privacy preserving data mining. *J. Cryptol.*, 15(3):177–206, 2002.

[11] R. J. A. Little and D. B. Rubin. *Statistical analysis with missing data*. John Wiley & Sons, Inc., USA, 1986.

[12] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *STOC*, pages 590–599, 2001.

[13] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *31st STOC*, pages 245–254, 1999.

[14] P. Paillier. Public-key cryptosystems based on composite degree residue classes. In *EUROCRYPT 99*, volume 1592 of *LNCS*, pages 223–238. Springer-Verlag, 1999.

[15] J. R. Quinlan. *Readings in knowledge acquisition and learning: automating the construction and improvement of expert systems*, chapter Induction of decision trees, pages 349–361. Morgan Kaufmann Publishers Inc., USA, 1993.

[16] A. C. Yao. How to generate and exchange secrets. In *27th FOCS*, pages 162–167, 1986.