

# Privacy-Preserving Imputation of Missing Data<sup>★</sup>

Geetha Jagannathan

*Stevens Institute of Technology  
Hoboken, NJ, 07030 USA*

Rebecca N. Wright

*Stevens Institute of Technology  
Hoboken, NJ, 07030 USA*

---

## Abstract

Handling missing data is a critical step to ensuring good results in data mining. Like most data mining algorithms, existing privacy-preserving data mining algorithms assume data is complete. In order to maintain privacy in the data mining process while cleaning data, privacy-preserving methods of data cleaning will be required. In this paper, we address the problem of privacy-preserving data imputation of missing data. Specifically, 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.

*Key words:* Data cleaning, Data imputation, Privacy-preserving protocols

---

---

<sup>★</sup> This work was supported by the National Science Foundation under Grant No. CCR-0331584. An extended abstract of this paper was published in the proceedings of the ICDM 2006 Workshop on the Privacy Aspects of Data Mining [16]

*Email addresses:* [gjaganna@cs.stevens.edu](mailto:gjaganna@cs.stevens.edu) (Geetha Jagannathan),  
[rwright@cs.stevens.edu](mailto:rwright@cs.stevens.edu) (Rebecca N. Wright).

## 1 Introduction

The cost per byte of secondary storage has fallen steeply over the years. Many organizations have taken advantage of this reduction in cost to create large databases of transactional and other data. These databases can contain information about retail transactions, customer and client information, geo-spatial information, web server logs, etc. These large data warehouses can be mined for knowledge that can improve the efficiency and efficacy of the organization. Additionally, the emergence of high speed networking and the ability to obtain better results by combining multiple sources of data have led to intense interest in distributed data mining.

However, due to human error or systemic reasons, large real-world data sets, particularly those from multiple sources, tend to be “dirty”—the data is incomplete, noisy, and inconsistent. If unprocessed raw data is used as input for data mining processes, the extracted knowledge is likely to be of poor quality as well. Therefore, *data cleaning* is an important preliminary step in data mining to improve the quality of the data and make the data more reliable for mining purposes. Data cleaning algorithms attempt to smooth noise in the data, identify and eliminate inconsistencies, and remove missing values or replace them with values imputed from the rest of the data,

Another concern in distributed settings is privacy. Specifically, due to the potential sensitivity of data, privacy concerns and privacy regulations often prevent the sharing of data between multiple parties. Privacy-preserving distributed data mining algorithms (e.g., [1, 19]) allow cooperative computation of data mining algorithms without requiring the participating organizations to reveal their individual data items to each other. Like most data mining algorithms, existing privacy-preserving data mining algorithms assume data is complete. Thus, in order to maintain privacy in the data mining process, privacy-preserving methods of data cleaning will also be required.

In this paper, we provide a first step in this direction—namely, a privacy-preserving solution to the data imputation problem. We note imputing missing values from “local” data alone may lead to biases. Performing imputation on all of the distributed data should result in more accurate predictions. (This seems particularly likely to be true in the case that it has been determined that the data mining should be carried out on the distributed data.) Our main result is 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. We also show privacy-preserving protocols for several other methods of data imputation.

To our knowledge, ours are the first results addressing privacy-preserving data cleaning. Such methods are a critical component of enabling privacy-preserving data mining algorithms to be used in practice, because if the data needs to be revealed in order to perform data cleaning, then privacy is lost, while if data cleaning is not performed, then the data mining results will be inaccurate.

### 1.1 *Related Work*

Several methods for dealing with missing values have been proposed. However, some of the simpler techniques have limited applicability or introduce bias into the data. The difficulty of handling missing data values depends on the randomness in the data [20]. If the distribution of the missing values is identical to those of recorded values, then almost any missing value replacement technique can be used without introducing bias in the data. Alternately, the probability of a row having a specific missing value for an attribute depends on the recorded values for the row, but not on the value of the missing data itself. In the hardest (and most general) case, the probability that a row has a specific value for an attribute depends on the value itself.

One of the easiest approaches to dealing with missing values is to simply delete those rows that have missing values. However, unless the missing values are distributed identically to the non-missing values, this produces poor quality data [26]. Another common technique for dealing with missing values is to create a new data value (such as “missing”) and use it to represent missing values. However, this has the unfortunate side-effect that data mining algorithms may try to use missing as a legal value, which is likely to be inappropriate. It also sometimes has the effect of artificially inflating the accuracy of some data mining algorithms on some data sets [11].

Data imputation is the process of replacing missing values with estimated values, which produces better results in subsequent data mining than either of the above methods. Imputation techniques range from fairly simple ideas (such as using the mean or mode of the attribute as the replacement for a missing value [6, 14]) to more sophisticated ones that use regression [3], Bayesian networks [7] and decision tree induction [17]. Using the mean or mode is generally considered a poor choice [20], as it distorts other statistical properties of the data (such as the variance) and does not take dependencies between attributes into account. Hot-deck imputation [10] fills in a missing value using values from other rows of the database that are similar to the row with the missing value. Hot-deck imputation has been performed with  $k$ -nearest neighbors algorithms [5, 2] and clustering algorithms [18]. In the clustering based technique, first, the rows that do not have missing data are partitioned into clusters. Each row with missing data is assigned to the cluster

“most similar” to it. The mean or the mode in the cluster of the attribute that is missing a value is substituted for the missing value.

Classification is generally considered the best method for imputing missing data [9]. For each attribute with missing values, the attribute with missing data is used as the dependent attribute (the class attribute), and the remaining attributes are used as the independent attributes for the data mining algorithm. The row with the missing attribute is used as an instance that requires prediction and then the predicted value is used for the missing value. While any classification or prediction algorithm can be used for imputation, the most commonly used methods are regression-based imputation and decision-tree-based imputation.

Regression imputation [3] imputes missing values with predicted values derived from a regression equation based on variables in the data set that contain no missing values. Regression assumes a specific relationship between attributes that may not hold for all data sets. The CART decision tree learning algorithm [4], which internally handles missing values, can be considered to use an implicit form of regression imputation. A privacy-preserving linear regression protocol is presented in [8]; this would be useful for imputation in cases where the missing values are linearly related with existing data values.

Decision-tree imputation uses a decision-tree based learning algorithm such as ID3 [25] or C4.5 [24] 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 [17, 9]. It has been observed in [17] and in [9] that single imputation using decision trees is more accurate than imputation based on clustering. In some cases, the decision tree construction can be *lazy* [11], in that only the needed path of the tree is constructed. This has an efficiency advantage because it avoids time spent on constructing the parts of the tree that are not needed.

We note that Lindell and Pinkas [19] provide a privacy-preserving algorithm for computing the ID3 tree for databases that are horizontally partitioned between two parties. However, we are unable to use their solution directly because it enables the parties to learn the computed decision tree (indeed, that is its goal). In our case, we want one or both of our participants to learn the imputed values determined by using the computed decision tree for classification, but we do not want the participants to learn the decision tree itself. Specifically, while our privacy-preserving data imputation solution uses ID3 trees, 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. (We do, however, make use of some of Lindell and Pinkas’s subprotocols.)

## 1.2 *Our Contributions*

Our main result is 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. We present two versions of the protocol that represent a privacy/efficiency trade-off. The first version is more efficient, but reveals the number of nodes traversed by the protocol in the undisclosed decision tree used for imputation. The second version, which has slightly increased communication and computation costs, does not leak any information beyond the computed imputed value. To our knowledge, this is the first paper addressing privacy-preserving methods of preprocessing data.

We also briefly describe secure protocols for data imputation based on other well-known imputation methods, namely, mean, mode, linear regression and clustering. Privacy-preserving imputation using other predictors such as Bayesian networks and nonlinear regression is left as future work.

We start in Section 2 by introducing some ideas and definitions used in the rest of the paper. In Section 3, we describe our basic privacy-preserving lazy decision tree imputation protocol. We analyze the communication complexity and privacy of the overall protocol in Section 4. In Section 5, we show how to obtain full privacy in our decision tree imputation protocol. In Section 6, we outline secure protocols for data imputation using the mean, the mode, linear regression, and clustering-based prediction.

## 2 Preliminaries

In this section, we describe the lazy decision tree algorithm on which our protocol is based, our privacy model, and some cryptographic primitives used in this paper.

### 2.1 *Lazy Decision Tree Algorithm*

We base our solution on a lazy decision tree algorithm that is a straightforward simplification of ID3. This algorithm lends itself to an efficient privacy-preserving distributed solution. Our algorithm fits the definition of Friedman et al. [11] for a generic lazy decision tree algorithm, though our algorithm

Algorithm Lazy Decision Tree

**Input:** A database  $D$  of labeled instances (with attributes  $R = \{A_1, \dots, A_m\}$ ), 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  $m$   
     $\text{Entropy}(A_i) = \text{Calc\_Split\_Entropy}(D, A_i)$
  - (b)  $A_{\min} = \text{Attribute with least entropy.}$
  - (c)  $D \leftarrow \{X \in D \mid X[A_{\min}] = I[A_{\min}]\}$
  - (d)  $R \leftarrow R - \{A_{\min}\}$
- (4) Go to Step 1

Fig. 1. Lazy Decision Tree Algorithm

differs significantly from the specific LazyDT algorithm presented in that paper. That algorithm is more complex, slower, and not easily amenable for conversion to a privacy-preserving protocol. The experiments in [11] indicate that LazyDT, on average, does have a small improvement in accuracy over 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.

Our algorithm (Figure 1) begins by using ID3’s information gain criterion to compute the attribute to be tested at the root of the tree. Those rows of the training data which match the test instance on the root attribute are filtered into the next iteration of the algorithm. A database  $D$  is said to be *pure* if all the transactions in  $D$  have the same class label. The following steps are repeated until the remaining instances are pure or all attributes have been exhausted:

1. Choose an attribute of high information gain, and
2. Winnow the training data to those instances that match the test data on the chosen attribute.

In either case, the algorithm predicts the class label of the test instance as the most frequently occurring class in the remaining set of instances.

The algorithm, as presented, does not directly calculate the attribute with the highest information gain. Instead, it calculates the attribute that results in the

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  $a_1, \dots, a_k$ .
2.  $\text{Entropy}(D, A) = - \sum_{j=1}^k \frac{|D(a_j)|}{|D|} \left( \sum_{i=1}^q \frac{p_{ji}}{|D(a_j)|} \log \frac{p_{ji}}{|D(a_j)|} \right)$

where

$D(a_j) =$  instances of  $D$  in which  $A$  has value  $a_j$ ,

$p_{ji} = \#$  instances of  $D(a_j)$  in which class label is  $c_i$ , for  $1 \leq i \leq q$

3. return  $\text{Entropy}(D, A)$

Fig. 2. Split Entropy

training (sub)set  $D$  having the least amount of entropy. In what follows, we assume that the class attribute takes values  $\{c_1, \dots, c_q\}$ . Assuming that an attribute  $A$  takes values in the set  $\{a_1, \dots, a_k\}$ , and that  $D_i =$  instances of  $D$  in which  $A$  has value  $a_i$ , for  $1 \leq i \leq k$ , the conditional entropy after splitting on attribute  $A$  is defined as

$$\text{Entropy}(D, A) = - \sum_{j=1}^k \frac{|D(a_j)|}{|D|} \left( \sum_{i=1}^q \frac{p_{ji}}{|D(a_j)|} \log \frac{p_{ji}}{|D(a_j)|} \right)$$

Here,  $p_{ji}$  is the number of instances of  $D(a_j)$  in which the class label is  $c_i$ , for  $1 \leq i \leq q$ .

## 2.2 Our Model

In our setting, two parties, Alice and Bob, respectively own databases  $D_A = (d_1, \dots, d_\ell)$  and  $D_B = (d_{\ell+1}, \dots, d_n)$  defined over a common set of attributes  $\{A_1, \dots, A_m\} \cup M$ . We use the notation  $\alpha \in$  Alice to indicate that Alice holds the value  $\alpha$ , and we use a similar notation for Bob. We describe our solution in the simplified scenario in which there is exactly one missing value to be learned. (In practice, the solutions for multiple missing values could be combined to make use of common sub problems for efficiency.) Specifically, suppose  $I \in$  Bob (not included in  $\{d_1, \dots, d_n\}$ ) 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.

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. The desired security condition in this model is that anything Alice or Bob learns from participating in the protocol could have been learned by simply giving them their initial input and final output. This is typically shown by showing a *simulator* for each party, showing that the party can simulate the interaction from their input and output alone. (For more detail, see [13].)

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,  $q$  denotes the number of values the class attribute can take and  $c$  denotes the maximum number of bits required to represent any encryption.

### 2.3 Cryptographic Primitives

**RANDOM SHARES.** 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  $a$  and Bob knows  $b$  and  $x$  can be recovered from  $a$  and  $b$ . This is known as 2-out-of-2 secret sharing. In our paper we use additive sharings and XOR sharings. We choose  $N$  to be a large prime, where the field  $F$  is isomorphic to  $\mathbb{Z}_N$ . We say Alice and Bob have random shares of  $x$  to mean that Alice knows a random value  $a \in F$  and Bob knows a random value  $b \in F$  such that  $(a + b) \bmod N \equiv x$ . By 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 remainder of the paper take place in  $F$ . We also assume suitable hardness assumptions under which the various cryptographic primitives described in this section are secure.

**HOMOMORPHIC ENCRYPTION.** Homomorphic encryption schemes allow certain computations on encrypted values. In particular, an encryption scheme is *additively homomorphic* if there is some operation  $\otimes$  on encryptions such that for all cleartext values  $a$  and  $b$ ,  $E(a) \otimes E(b) = E(a + b)$ . Our solutions make use of an semantically secure additively homomorphic encryption scheme [23].

SECURE CIRCUIT EVALUATION PROTOCOL. Yao’s two-party secure circuit-evaluation protocol [27] allows two parties holding inputs  $a$  and  $b$  to privately compute any function  $f(a, b)$  without revealing  $a$  and  $b$  to each other. In theory, this could solve any distributed two-party privacy-preserving data mining problem. However, as a practical matter, the circuits for even megabyte-sized databases would be intractably large. We will make use of Yao’s protocol for private computation in several cases, but only on a small number of small inputs. We note that the performance of Yao’s protocol is heavily dependent in the performance of an underlying cryptographic primitive called *oblivious transfer*.

PURITY COMPUTATION PROTOCOL. In the purity computation protocol [19], Alice has a vector of values  $X = (x_1, \dots, x_n)$  and Bob has a vector of values  $Y = (y_1, \dots, y_n)$ . The protocol outputs  $c$  if  $x_1 = \dots = x_n = y_1 = \dots = y_n = c$ , or  $\perp$  otherwise. Let  $\perp$  be the symbol that denote the fact the a party has more than one class. Alice and Bob input either  $\perp$  or  $c$ . They run a secure protocol for equality testing with their inputs and output a value that is either  $\perp$  to indicate that  $D$  is not pure or  $c$  which is the majority label and is the missing value in our case.

SECURE  $x \ln(x)$  PROTOCOL. When Alice and Bob have random shares of  $x$  denoted as  $v_1$  and  $v_2$  respectively, the secure  $x \ln(x)$  protocol [19] computes the shares of  $x \ln(x)$  as  $w_1$  and  $w_2$  for Alice and Bob respectively.

PRIVATE INDIRECT INDEX PROTOCOL. In a private indirect index (PIX) protocol [21], Bob has a vector of values  $X = (x_1, \dots, x_n)$ . 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 [21] outputs an XOR-sharing of  $x_i$ , it can be easily modified to give an additive sharing of the output. This protocol makes use of one invocation of 1-out-of- $n$  oblivious transfer, which we write as  $\text{OT}_1^n$ . 1-out-of- $n$  oblivious transfer [22] is a two party protocol where the sender has  $n$  inputs  $\{x_1, \dots, x_n\}$  and the receiver has an input  $j \in \{1, \dots, n\}$ . At the end of the protocol, the receiver learns  $x_j$  and no other information and the sender learns nothing.

SECURE SCALAR PRODUCT PROTOCOL. In a private scalar product protocol SPP protocol, Alice has a vector  $X = (x_1, \dots, x_n)$  and Bob has a vector  $Y = (y_1, \dots, y_n)$ . They wish to securely compute the scalar product as  $X \cdot Y \equiv s_A + s_B \pmod N$ , where  $s_A$  and  $s_B$  are the random shares of Alice and Bob respectively.

### 3 Privacy-Preserving Imputation Based on Lazy Decision Trees

We now describe our main privacy-preserving data imputation protocol. It is based on the lazy decision tree algorithm described in Section 2.1. In this section, we describe a protocol that has a small privacy leak: it reveals the number of nodes traversed by the protocol in the undisclosed decision tree used for imputation. In Section 5, we show how this leak can be removed at a slightly increased cost of communication and computation.

Recall the definitions of  $D_A$ ,  $D_B$  and  $I$  given in Section 2.2. Bob wishes to compute the missing value  $I(M)$  by applying the lazy decision tree imputation on  $D = D_A \cup D_B$ . At the end of the protocol, Bob learns only the missing value  $I(M)$  and Alice learns nothing. Both parties learn nothing else. (For missing values in Alice’s database, they would reverse roles from what we describe here.) They learn nothing about the path of the decision tree including the attributes on each node and the value taken by the attributes along the path that leads to the missing value. In this section, we assume that the class attribute  $M$  takes the values  $\{c_1, \dots, c_q\}$ .

#### 3.1 Our Basic Protocol

We rephrase the basic steps of the lazy decision tree algorithm from Section 2.1 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 stringent privacy requirements prohibit the protocol from revealing any information other than the final output. In particular, this implies that none of the attributes that have been chosen along the way may be revealed to either party. Further, the subset of the instances that match  $I$  on the chosen attributes cannot be revealed either. 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 would have  $s_A$  and Bob would have  $s_B$  such that  $s_A \oplus s_B = s$ . To satisfy the second condition, the protocol

Protocol Private Lazy Decision Tree Data Imputation

**Input:** A database  $D = D_A \cup D_B$  of labeled instances (with attributes  $\{A_1, \dots, A_m\} \cup M$ ), where Alice owns  $D_A = (d_1, \dots, d_\ell)$  and Bob owns  $D_B = (d_{\ell+1}, \dots, d_n)$  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  $m$ , Alice and Bob compute random shares of  $\text{Entropy}(D, A_i)$  as  $\text{Ent}_i^A + \text{Ent}_i^B \equiv \text{Entropy}(D, A_i) \pmod{N}$ .
  - (b) Using Yao's protocol, Alice and Bob compute  $\min_A$  and  $\min_B$  such that  $\min_A \oplus \min_B = \min$  where  $\text{Ent}_{\min}^A + \text{Ent}_{\min}^B \equiv \text{minimum}\{\text{Ent}_i^A + \text{Ent}_i^B\}, 1 \leq i \leq m$ .
3. for  $j = 1$  to  $m - 1$ 
  - (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 a PIX protocol 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  $\alpha, \beta$  to obtain two bit vectors  $P = (p_1, \dots, p_n) \in$  Alice and  $Q = (q_1, \dots, q_n) \in$  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  $m$ ,  $\text{Ent}_i^A + \text{Ent}_i^B = \text{Secure\_Ent\_Comp}(D_j, P, Q, A_i)$
    - Using Yao's protocol, Alice and Bob compute  $\min_A$  and  $\min_B$  such that  $\min_A \oplus \min_B = \min$  where  $\text{Ent}_{\min}^A + \text{Ent}_{\min}^B \equiv \text{minimum}\{\text{Ent}_i^A + \text{Ent}_i^B\}, 1 \leq i \leq m$ .
4. Bob outputs the majority label as the missing value using the secure majority computation protocol (see Section 3.6)

Fig. 3. Private Lazy Decision Tree Imputation Protocol

uses a randomly shared bit-vector representation of any  $D' \subseteq D$ . That is, Alice and Bob have, respectively, bit-vectors  $(p_1, \dots, p_n)$  and  $(q_1, \dots, q_n)$  such that for any  $d_i \in D$ :

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

Protocol `Secure_Ent_Comp`

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

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

- (1) Let  $A$  take values  $a_1, \dots, a_k$ .
- (2) for  $j = 1$  to  $k$ 
  - (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  $1 \leq i \leq n$ ,
$$R_i \oplus S_i = \begin{cases} 1 & \text{if } p_i \oplus q_i = 1 \text{ and } d_i[A] = a_j \\ 0 & \text{otherwise} \end{cases}$$

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  $p_{j\ell} = \#$  instances of  $D(a_j)$  in which class label is  $c_\ell$  for  $1 \leq \ell \leq q$ . Alice and Bob engage in the protocol described in Section 3.4.2 and output random shares of  $p_{j\ell}$ .
  - (d) They use the secure  $x \log x$  protocol of [19] to compute random shares of  $|D(a_j)| \log |D(a_j)|$ ,  $p_{j\ell} \log p_{j\ell}$  for  $1 \leq \ell \leq q$ .
  - (e) They compute random shares of  $\text{sum}_j = |D(a_j)| \log |D(a_j)| - \sum_{\ell=1}^q (p_{j\ell} \log p_{j\ell})$ .
- (3) Alice and Bob compute random shares of entropy for the attribute  $A$  by adding the shares obtained in Step 2e.

Fig. 4. **Secure Split Entropy**

In addition, 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 3. 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 the purity checking protocol [19] as described in Section 2.3.

To compute the root of the lazy decision tree, Alice and Bob compute random shares of the entropy for every attribute  $\{A_1, \dots, A_m\}$ . At the root level com-

puting the shares of the entropy is straightforward. Recall that  $a_1, \dots, a_k$  is the domain of attribute  $A$ . The conditional entropy of a database  $D$  (explained in Section 2.1) with respect to the attribute  $A$  can be rewritten as

$$\text{Entropy}(D, A) = - \sum_{j=1}^k \left( \sum_{\ell=1}^q p_{j\ell} \log(p_{j\ell}) - |D(a_j)| \log(|D(a_j)|) \right)$$

where  $p_{j\ell}$ , for  $1 \leq \ell \leq q$  and  $D(a_j)$  are as explained in Figure 2. Because the database is horizontally partitioned between Alice and Bob, they compute the shares of  $p_{j\ell}$ , and  $D(a_j)$  independently. Using the secure protocol that computes a random sharing of  $x \log x$  [19] given a sharing of  $x$ , Alice and Bob jointly compute a random sharing of  $\text{Entropy}(D, A_i)$  for each attribute  $i = 1, \dots, m$ .

Alice has a vector  $(\text{Ent}_i^A)$  of entropy shares and Bob, correspondingly, has  $(\text{Ent}_i^B)$ , for  $i = 1, \dots, m$  such that  $\text{Ent}_i^A + \text{Ent}_i^B \equiv \text{Entropy}(D, A_i) \pmod{N}$ . The index of the attributes that yields the least entropy is computed as random shares ( $\min_A \in \text{Alice}$  and  $\min_B \in \text{Bob}$  such that  $\min = \min_A \oplus \min_B$ ) between Alice and Bob using Yao's protocol. Note that either  $\text{Ent}_i^A + \text{Ent}_i^B = \text{Entropy}(D, A_i)$  or  $\text{Ent}_i^A + \text{Ent}_i^B = \text{Entropy}(D, A_i) + N$ . The circuit first computes  $\text{Ent}_i^A + \text{Ent}_i^B$  and if it is greater than  $N - 1$  it subtracts  $N$ . It then selects the attribute with the minimum entropy. We emphasize that our privacy criterion insists that the attributes that have been chosen at various stages in the path of the lazy decision tree computation should not be revealed to either party. Hence in our protocol we maintain them as random shares between Alice and Bob.

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  $j$  of the lazy decision tree, 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 the attribute index  $s$  is randomly shared between Alice and Bob. Alice and Bob compute a random sharing of  $I[A_s]$  using a PIX protocol. Since  $D_j$  should be unknown to either party, we store the set in the form of two bits  $p_i \in \text{Alice}$  and  $q_i \in \text{Bob}$  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 using the secure protocol that checks if  $D_j$  is pure and returns the majority value. It is important to observe that since neither party knows which of their instances are in  $D_j$ , we cannot use here the purity check in [19]. We provide an alternate purity checking protocol in Section 3.5. If  $D_j$  is not pure, Alice and Bob engage in the secure entropy computation protocol for the split  $D_j$  for each of the attributes  $\{A_1, \dots, A_m\}$ , and computes the random shares of the split entropy  $\text{Entropy}(D, A_i)$ . (ie.,  $\text{Ent}_i^A + \text{Ent}_i^B \equiv \text{Entropy}(D, A_i) \pmod{N}$ ) The attribute with the least entropy is computed using the Yao's protocol. 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.

Figure 3 presents the overall privacy-preserving lazy decision tree imputation protocol. The subprotocols used in this protocol are the secure split protocol (Section 3.3), a secure protocol that computes split entropy (Figure 4), a secure protocol that checks if the database is pure (Section 3.5) and a majority computation protocol (Section 3.6). In Section 3.4, we present secure subprotocols for split entropy computation.

### 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 which is 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  $A$ . Let us assume that the attribute  $A$  takes values  $a_1, \dots, a_k$ . In the decision tree computation, the entropy after splitting the database  $D'$  on attribute  $A$  is given by

$$-\sum_{j=1}^k \left( \sum_{\ell=1}^q p_{j\ell} \log(p_{j\ell}) - |D(a_j)| \log(|D(a_j)|) \right)$$

For every  $1 \leq j \leq k$ , Alice and Bob communicate with each other to compute

a set  $D(a_j) \subset D'$ . The set  $D(a_j)$  is represented by two bit vectors  $R = (R_1, \dots, R_n) \in \text{Alice}$  and  $S = (S_1, \dots, S_n) \in \text{Bob}$ , where

$$R_i \oplus S_i = \begin{cases} 1 & \text{if } p_i \oplus q_i = 1, d_i[A] = a_j, 1 \leq i \leq n \\ 0 & \text{otherwise} \end{cases}$$

This is achieved using Yao's secure circuit evaluation protocol.  $|D(a_j)|$  and  $p_{j\ell}$ , for  $1 \leq \ell \leq q$ ,  $1 \leq j \leq k$ , are computed as random shares between Alice and Bob using the protocols explained in Section 3.4. Random shares of the terms  $p_{j\ell} \log p_{j\ell}$ , for  $1 \leq \ell \leq q$  are computed using the secure computation of  $x \log x$  protocol given by [19]. We present the entropy computation protocol in Figure 4.

The size of the circuit in Step 2a is a constant and this circuit evaluation happens  $n$  times and hence the communication complexity is  $O(n)$ . Step 2b involves two parties jointly computing random shares of  $|D(a_j)|$  using the protocol described in Section 3.4.1. The communication complexity is given by  $O(cn)$ . The computational complexity is also given by the cost of the SPP protocol. Step 2c involves computing random shares of the number of instances of  $D(a_j)$  that has labels  $\{c_1, \dots, c_q\}$  using the protocol described in Section 3.4.2. The communication complexity is given by  $O(cnq)$ . Its computational complexity is given by the cost of the SPP protocol. The  $x \log x$  protocol given by [19] has a communication complexity of  $O(|S| \log n)$  bits and a computational complexity of  $O(\log n) \text{ OT}_1^2$ , where  $|S|$  is the length of the key of a pseudorandom function used in the  $x \log x$  protocol.

The total communication complexity of the **Secure Split Entropy** protocol is given by  $O(cknq) + O(qk|S| \log n)$ . The total computation complexity is given by  $O(qk \log n) \text{ OT}_1^2$  and  $O(qk)$  executions of the SPP protocol that involves  $O(qnm)$  encryptions and  $O(qk)$  decryptions for Alice and  $O(nqk)$  exponentiations and  $O(qk)$  encryption for Bob.

The **Secure Split Entropy** protocol securely computes random shares of the entropy after splitting  $D'$  on the attribute  $A$ . The Yao's circuit evaluation protocol securely computes the subset  $D(a_j)$  of the database whose instances take a specific value for the attribute  $A$ . This subset is stored as a random sharing between the two parties. We have shown in Section 3.3 that the protocols used to compute the size of the split as random shares between the two parties are secure and do not leak any information. Finally, the protocol used to compute  $x \log x$  [19] as random shares is also secure. The output of the **Secure Split Entropy** protocol is a random sharing of the entropy after splitting  $D'$  on the attribute  $A$ . This is the only information available to both parties when a trusted third party is used.

### 3.3 Secure Split Protocol

The secure split protocol is used to find if a given record is in a subset  $D' \subseteq D$  and attribute  $A_i$  takes the value  $\alpha$  in that record. The result of this test (0 or 1) is randomly shared between Alice and Bob. Here inputs  $\alpha$  and  $i$  are shared between Alice and Bob as  $\alpha \equiv a + b \pmod{N}$  and  $i = i_1 \oplus i_2$ , where  $a$  and  $i_1$  are known to Alice and  $b$  and  $i_2$  are known to 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 (the case where Bob owns the record is similar), Alice's inputs to the protocol are the record  $(v_1, \dots, v_m)$ ,  $a$ ,  $i_1$  and  $p$ . Bob's inputs are  $b$ ,  $i_2$  and  $q$ .

At the end of this protocol, Alice and Bob receive bits  $b_1$  and  $b_2$  respectively such that

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

In other words, the fact that a given attribute has a specific value in a given record is available to both Alice and Bob only as random shares. This protocol has two stages. In the first stage, Alice and Bob use a PIX protocol and output  $v$  and  $w$ , respectively, where  $v + w \equiv v_i \pmod{N}$ . 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

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

The communication and computational complexity of the first stage is that of one invocation of  $\text{OT}_1^m$ . The second stage of the protocol involves one comparison which is done using Yao's circuit evaluation protocol [27] and this requires constant bits of communication. The total communication and computational complexity for one invocation of the secure split computation protocol is the same as one invocation of  $\text{OT}_1^m$ .

The split computation protocol described in Section 3.3 is a secure protocol for two semi-honest parties. The first stage of the protocol uses a secure PIX protocol [21]. Yao's circuit evaluation protocol is also secure and does not leak

any information.

### 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  $X = (x_1, \dots, x_n)$  and  $Y = (y_1, \dots, y_n)$  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 the  $|D'|$ . We present this protocol in Figure 5.

The only communication that happens between the two parties is during the execution of the SPP protocol using two vectors of size  $n$ . The communication complexity of the SPP protocol [12] is  $O(cn)$ . The computational complexity involves  $n$  encryptions and one decryption for Alice and  $n$  exponentiations and one encryption for Bob. The SPP protocol is secure and it outputs no information other than the random shares to the two parties. Hence this protocol is secure.

#### 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 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  $c$  ( $c \in \{c_1, \dots, c_q\}$ ).  $D'$  is represented by the bit vectors  $X = (x_1, \dots, x_n)$  and  $Y = (y_1, \dots, y_n)$  known to Alice and Bob respectively, where

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

Protocol Total\_Record\_Split

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

**Output:** Alice and Bob output  $\alpha$  and  $\beta$ , respectively, such that  $\alpha + \beta \equiv |D'| \pmod{N}$ , 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  $\beta = \delta - 2\lambda$ .

Fig. 5. **Secure Computation of  $|D'|$**

The total number of records in the split with a given class value is given by the number of the records that satisfy the relation  $(x_i \oplus y_i) \wedge w_i = 1$  for  $1 \leq i \leq n$ . This relation can be rewritten as  $(\overline{x_i w_i}) y_i \vee (x_i w_i) \overline{y_i} = 1$ . Hence, the number of records in the split is given by the sum of the records satisfying  $(\overline{x_i w_i}) y_i = 1$  and the ones satisfying  $(x_i w_i) \overline{y_i} = 1$ . It is computed as random shares between Alice and Bob using a secure SPP protocol [12]. We present this protocol in Figure 6.

The communication and computational complexity are the same as that of the SPP protocol. The security of this protocol follows from the security of the scalar product protocol.

### 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  $P = (p_1, \dots, p_n) \in$  Alice and  $Q = (q_1, \dots, q_n) \in$  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  $|D'_i|$ , for  $1 \leq i \leq q$  using the secure protocol described in Section 3.4.2. Here  $|D'_i|$  denotes the number of records in  $D'$  where the class attribute takes the value  $c_i$ . Let  $\alpha_i^A$  and  $\alpha_i^B$  denote Alice's and Bob's share of  $|D'_i|$ , for  $1 \leq i \leq q$  respectively. Alice and Bob

Protocol Total\_Record\_Split\_given\_Class\_Value

**Input:** A database  $D$  of labeled instances where Alice owns  $D_A = (d_1, \dots, d_\ell)$  and Bob owns  $D_B = (d_{\ell+1}, \dots, d_n)$ ,  $D' \subseteq D$  represented by bit vectors  $X = (x_1, \dots, x_n)$  and  $Y = (y_1, \dots, y_n)$  belonging to Alice and Bob respectively such that for  $1 \leq i \leq n$ ,  $x_i \oplus y_i = 1$  if  $d_i \in D'$ , and 0 otherwise.

**Output:** Alice and Bob output  $\alpha$  and  $\beta$ , respectively, such that  $\alpha + \beta \equiv |D'_c| \pmod{N}$ , where  $|D'_c|$  is the total number of records for which  $x_i \oplus y_i = 1$  and  $M$  takes the value  $c$ .

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

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

2. Bob computes a vector  $W = (w_{\ell+1}, \dots, w_n)$  such that, for  $\ell + 1 \leq i \leq n$ ,

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

3. Alice and Bob run SPP protocol with inputs

- $(\overline{x_1}w_1, \dots, \overline{x_\ell}w_\ell)$  and  $(y_1, \dots, y_\ell)$ , respectively, to obtain shares  $\alpha_1$  and  $\beta_1$ , respectively.
- $(x_1w_1, \dots, x_\ellw_\ell)$  and  $(\overline{y_1}, \dots, \overline{y_\ell})$ , respectively, to obtain shares  $\alpha_2$  and  $\beta_2$  respectively.
- $(\overline{x_{\ell+1}}w_{\ell+1}, \dots, \overline{x_n}w_n)$  and  $(y_{\ell+1}, \dots, y_n)$ , respectively, to obtain shares  $\gamma_1$  and  $\delta_1$ , respectively.
- $(x_{\ell+1}w_{\ell+1}, \dots, x_nw_n)$  and  $(\overline{y_{\ell+1}}, \dots, \overline{y_n})$ , respectively, to obtain shares  $\gamma_2$  and  $\delta_2$  respectively.

4. Alice outputs  $\alpha = \alpha_1 + \alpha_2 + \gamma_1 + \gamma_2$  and Bob outputs  $\beta = \beta_1 + \beta_2 + \delta_1 + \delta_2$ .

Fig. 6. **Secure Computation of  $|D'_c|$**

use Yao's protocol to check if there exists a  $j$  such that  $\alpha_j^A + \alpha_j^B \neq 0 \pmod{N}$  and  $\alpha_i^A + \alpha_i^B \equiv 0 \pmod{N}$  for every  $i \neq j$ . If so Bob outputs  $c_j$ . Otherwise he outputs  $\perp$ .

Yao's protocol for the comparison described above requires  $O(q)$  bits of communication. The communication and the computational complexity is dominated by the protocol computing  $|D'_i|$ , for  $1 \leq i \leq q$ . The complexity are the same as described in Section 3.4.2. The protocol that is used to compute  $\alpha_i^A$  and  $\alpha_i^B$  returns only random shares to Alice and Bob. Yao's protocol is secure and does not leak any information. Hence at the end of the protocol, Bob gets the missing value if the data set is pure.

### 3.6 Secure Protocol for Majority Computation

This protocol is similar to the protocol (Section 3.5) that checks if  $D$  is pure. In the first stage, Alice and Bob compute random shares of  $|D_i| \equiv \alpha_i^A + \alpha_i^B \pmod N$ , where  $\alpha_i^A \in \text{Alice}$ , and  $\alpha_i^B \in \text{Bob}$ , for  $1 \leq i \leq q$  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  $\alpha_i^A$ , and Bob’s input as  $\alpha_i^B$ . Bob outputs the majority label as  $c_j$  if  $c_j = \max\{c_1, \dots, c_q\}$ . The complexities and the privacy of this protocol are the same as explained in Section 3.5.

## 4 Performance Analysis

In this section, we analyze the complexity and the privacy of the overall privacy-preserving lazy decision tree imputation algorithm given in Figure 3. Recall  $n$  denotes the size of the database,  $m$  denotes the number of attributes,  $k$  denotes the maximum number of values any attribute can take, and  $c$  denotes the maximum number of bits required to represent any encryption.

**COMPLEXITY.** Alice and Bob communicate with each other to compute the attribute that has the maximum information gain at each level of the path. However, neither party should know what attribute was selected at each node. This is achieved using the protocols listed in Section 3. The communication complexity is dominated by  $O(cnm^2kq) + O(m^2kqS \log n)$ , where  $S$  is the length of the key of the pseudo-random function used in the  $x \log x$  protocol [19]. The total computation complexity is given by  $O(m^2kq \log n) \text{OT}_1^2 + O(m) \text{OT}_1^m + O(m^2kq)$  executions of the SPP protocol (which involves  $O(m^2kqn)$  encryptions and  $O(m^2kq)$  decryptions for Alice and  $O(m^2kqn)$  exponentiations and  $O(m^2kq)$  encryptions for Bob).

**PRIVACY.** In terms of privacy, we compare this protocol with the one that uses the trusted third party (TTP). When a TTP is used, the two parties send their shares of data to the TTP. The TTP computes the missing value and sends it to Bob, and Alice gets nothing. Both parties receive no other information other than Bob receiving the missing value.

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 only available as random shares to Alice and Bob. This is achieved through the protocols of Section 3. As discussed in Section 3, these protocols do not leak any information. 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 in the TTP solution. In Section 5, we describe a modification of the **Private Lazy Decision Tree Data Imputation** protocol, which prevents even the number of attributes in the path from being learned by Alice and Bob.

## 5 Enhancing Privacy

As just discussed, the protocol of Section 3 has a minor leak—it reveals the number of nodes in the evaluation path of the decision tree used in the imputation process. In this section, we outline a modified version of the protocol which eliminates this leak, thus achieving the same level of privacy as in the TTP model.

The modified version of the protocol uses two additional variables. One of the variables is called `is_pure`, and the other is called `majority_class`. We treat `is_pure` as a Boolean (with 0 = `false` and 1 = `true`). Each of these variables is randomly shared between Alice and Bob, respectively, as `is_pureA` and `is_pureB`, and `majority_classA` and `majority_classB`. At the end of each iteration, the protocol ensures that `is_pureA ⊕ is_pureB = is_pure` and `majority_class ≡ majority_classA + majority_classB mod N`. Initially, `is_pure` is set to `false` (by setting `is_pureA = false` and `is_pureB = false`). If `is_pure` becomes `true` at the end of some iteration, it means that the protocol has determined that the data set  $D$  at the beginning of that iteration is pure (all rows have the same class label). At the end of each iteration, the protocol ensures that `majority_class` contains the most frequently occurring class label in data set  $D$ .

In the modified version of our protocol, `is_pureA`, `is_pureB`, `majority_classA` and `majority_classB` are supplied as additional inputs to the subprotocol that checks if  $D$  is pure (see Section 3.5). The modified version checks if `is_pure` is `false`. If so, the protocol computes `is_pure` and `majority_class`, in the form of random shares for Alice and Bob. On the other hand, if `is_pure` is `true` when the subprotocol is invoked, `is_pure` and `majority_class` are not recomputed. Instead, a new random sharing of each is computed and sent to Alice and Bob. It is important to realize that the new values of `is_pureA` and `is_pureB` (respectively `majority_classA` and `majority_classB`) at the end of the execution of the subprotocol may be different from their values at the start of the subprotocol, even if the value of `is_pure` (respectively `majority_class`) does not change in the subprotocol.

The subprotocol for computing the majority class (described in Section 3.6) is also modified. The values of `is_pureA`, `is_pureB`, `majority_classA` and `majority_classB`

are supplied as additional inputs to the modified version. The subprotocol first checks if `is_pure` is `false`. If so, the computation for the `majority_class` proceeds as usual. The value of `majority_class` is randomly shared as `majority_classA` and `majority_classB`. On the other hand, if `is_pure` is `true`, the computation of the majority class is skipped. Instead, `majority_classA + majority_classB mod N` is used in place of `majority_class`, and this is again randomly shared as `majority_classA` and `majority_classB`.

Given these modifications, the new version of the main protocol runs as many iterations as there are attributes. That is, the iteration of the main loop in the protocol does not end when a pure set  $D$  is obtained. At the end of all iterations, Alice sends the value of `majority_classA` to Bob who then uses `majority_classA + majority_classB mod N` as the imputed value.

## 6 Other Imputation Protocols

In this section, we briefly describe other private data imputation protocols. As explained in Section 1.1, however, the results of the decision tree solution are likely to be better in most settings, Recall from Section 2.2 that two parties, Alice and Bob, own databases  $D_A = (d_1, \dots, d_\ell)$  and  $D_B = (d_{\ell+1}, \dots, d_n)$ , respectively, defined over a common set of attributes  $\{A_1, \dots, A_k\} \cup M$ . The instance  $I \in$  Bob has a missing value for the attribute  $M$ . Let  $\alpha_1, \dots, \alpha_n$  denote the values of the attribute  $M$  for the instances  $d_1, \dots, d_n$ .

**MEAN.** Alice computes  $a = \sum_{j=1}^{\ell} \alpha_j$  and sends  $a$  to Bob. Bob computes  $b = \sum_{j=(\ell+1)}^n \alpha_j$  and hence computes the missing value  $I(M)$  as  $(a + b)/n$ . Even though this protocol satisfies the privacy requirements described in Section 2.2, Bob learns Alice's sum of values of the attribute  $M$ .

**MODE.** Let us assume that the attribute  $M$  takes values from the domain  $\{c_1, \dots, c_p\}$ . Alice computes the frequencies of  $\{c_1, \dots, c_p\}$  from her database  $D_A$  as  $\alpha_1, \dots, \alpha_p$  and Bob computes the frequencies from his database  $D_B$  as  $\beta_1, \dots, \beta_p$ . Alice and Bob use Yao's protocol with inputs  $\alpha_1, \dots, \alpha_p$  and  $\beta_1, \dots, \beta_p$  and Bob gets as output the index  $i$  such that  $\alpha_i + \beta_i = \text{minimum } \{\alpha_j + \beta_j\}$ , for  $1 \leq j \leq p$ . Bob outputs  $c_i$  as the missing value.

**LINEAR REGRESSION.** We describe the protocol for two variables. The extension to multiple regression is straightforward. Suppose  $x$  is an independent variable and  $y$  is the variable that has a missing value. Linear regression involves fitting the straight line  $y = mx + b$ , where

$$m = \frac{\sum_{i=1}^n (x_i y_i) - (\sum_{i=1}^n x_i \sum_{i=1}^n y_i) / n}{\sum_{i=1}^n x_i^2 - (\sum_{i=1}^n x_i)^2 / n}$$

$$b = \frac{\sum_{i=1}^n x_i}{n} - m \cdot \frac{\sum_{i=1}^n y_i}{n}.$$

Alice and Bob can securely compute shares of  $m$  and  $b$  and using these shares Bob can compute the missing value.

**CLUSTERING.** Alice and Bob use a secure clustering protocol (such as the one in [15]) to compute shares of  $m$  cluster centers, where  $m$  is a user-specified parameter. Let  $C_i^A = (a_{i1}^A, \dots, a_{ik}^A, p_i^A)$  and  $C_i^B = (a_{i1}^B, \dots, a_{ik}^B, p_i^B)$ , for  $1 \leq i \leq m$ , denote the shares of the  $m$  cluster centers for Alice and Bob, respectively. Let the instance  $I = (x_1, \dots, x_k, x)$ , where  $x$  denotes the missing value.

Alice and Bob jointly and securely compute the shares of the distance between  $I$  and each of the  $m$  cluster centers using only the first  $k$  coordinates. Let  $Z^A = (z_1^A, \dots, z_m^A)$  and  $Z^B = (z_1^B, \dots, z_m^B)$  denote Alice’s and Bob’s shares of the distances respectively. They use Yao’s secure circuit evaluation protocol to compute random shares of the index  $i$  such that  $z_i^A + z_i^B = \text{minimum} \{z_j^A + z_j^B\}$ , for  $1 \leq j \leq m$ . Alice’s share is  $i_1$  and Bob’s share is  $i_2$  such that  $i = i_1 \oplus i_2$ . Bob should output the missing value as  $p_i^A + p_i^B$ , where  $p_i^A$  (respectively,  $p_i^B$ ) is Alice’s share (respectively, Bob’s share) of the missing attribute in cluster  $C_i$ . One possible way to do this would be to reveal  $i$  to both parties—Alice could then send  $p_i^A$  to Bob so that he can compute the missing value. However, Bob would then learn the cluster from which the missing value is computed, which would be a leak according to our privacy definitions. This leak can be prevented by using a different approach. Alice and Bob invoke a PIX protocol twice, once for the vector  $(p_1^A, \dots, p_m^A)$  and the other for the vector  $(p_1^B, \dots, p_m^B)$ . In both invocations they supply  $i_1$  and  $i_2$  as input. This gives them random shares of  $p_i^A$  and of  $p_i^B$ . Alice adds her shares and sends them to Bob for him to compute the missing value.

## 7 Conclusion

We have presented a privacy-preserving protocol for filling in missing data values in horizontally partitioned data using a lazy decision tree imputation algorithm. The participants of the protocol learn only the imputed values; the computed decision tree is not learned by either party. We have also sketched protocols for other data imputation methods. Future directions include privacy-preserving solutions for other data imputation methods, such

as networks and nonlinear regression, extending the decision tree imputation algorithm to the multiparty case and involving malicious adversaries.

## References

- [1] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *ACM SIGMOD*, pages 439–450, May 2000.
- [2] G. E. A. P. A. Batista and M. C. Monard. An Analysis of Four Missing Data Treatment Methods for Supervised Learning. *Applied Artificial Intelligence*, 17(5):519–533, 2003.
- [3] J-F. Beaumont. On regression imputation in the presence of nonignorable nonresponse. In *The Survey Research Methods Section*, pages 580–585. ASA, 2000.
- [4] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Chapman and Hall, 1984.
- [5] J. Chen and J. Shao. Nearest neighborhood imputation for survey data. *J. Official Statistics*, 16(2):113–131, 2000.
- [6] P. Clark and T. Niblett. The CN2 induction algorithm. *Mach. Learn.*, 3(4):261–283, 1989.
- [7] L. Coppola, M. Di 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.
- [8] W. Du, Y. Han, and S. Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *Proc. of the Fourth SIAM International Conference on Data Mining*, pages 222–233, 2004.
- [9] 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.
- [10] B. L. Ford. *Incomplete data in sample surveys*, chapter An overview of hot-deck procedures. Academic Press, 1983.
- [11] 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.
- [12] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikainen. On secure scalar product computation for privacy-preserving data mining. In *7th ICISC*, 2004.
- [13] O. Goldreich. *Foundations of Cryptography, Vol II*. Cambridge University Press, 2004.
- [14] M. Hu, S.M. Salvucci, and M.P. Cohen. Evaluation of some popular imputation algorithms. In *The Survey Research Methods Section of the ASA*, pages 308–313, 1998.
- [15] G. Jagannathan, K. Pillaipakkamatt, and R. Wright. A new privacy-preserving distributed  $k$ -clustering algorithm. In *Proc. of the 6th SIAM*

- International Conference on Data Mining*, pages 492–496, 2006.
- [16] G. Jagannathan and R. N. Wright. Privacy-preserving data imputation. In *Proc. of the ICDM Int. Workshop on Privacy Aspects of Data Mining*, pages 535–540, 2006.
  - [17] K. Lakshminarayan, S. A. Harp, and T. Samad. Imputation of missing data in industrial databases. *Applied Intelligence*, 11(3):259–275, 1999.
  - [18] R. C. T. Lee, J. R. Slagle, and C. T. Mong. Towards automatic auditing of records. *IEEE Trans. Software Eng.*, 4(5):441–448, 1978.
  - [19] Y. Lindell and B. Pinkas. Privacy preserving data mining. *J. Cryptol.*, 15(3):177–206, 2002.
  - [20] R. J. A. Little and D. B. Rubin. *Statistical analysis with missing data*. John Wiley & Sons, Inc., USA, 1986.
  - [21] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *STOC*, pages 590–599, 2001.
  - [22] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *31st STOC*, pages 245–254, 1999.
  - [23] P. Paillier. Public-key cryptosystems based on composite degree residue classes. In *EUROCRYPT 99*, volume 1592 of *LNCS*, pages 223–238. Springer-Verlag, 1999.
  - [24] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
  - [25] 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.
  - [26] W. Sarle. Prediction with missing inputs. In *Proc. of the Fourth Joint Conference on Information Sciences*, pages 399–402, 1998.
  - [27] A. C.-C. Yao. How to generate and exchange secrets. In *27th FOCS*, pages 162–167, 1986.