Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering 65 (2008) 40–56
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Privacy-preserving imputation of missing data ☆

Geetha Jagannathan, Rebecca N. Wright *

Stevens Institute of Technology, Hoboken, NJ 07030, USA

Received 5 June 2007; accepted 5 June 2007
Available online 18 July 2007

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 are required. In this paper, we address the problem of privacy-preserving data imputation of missing data. 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.

Keywords: Data cleaning; Data imputation; Privacy-preserving protocols

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. The resulting data warehouses can be mined for knowledge that can improve the efficiency and efficacy of organizational processes. Further, 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 and 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*, which seeks to improve the quality of the data and make the data more reliable for mining purposes, is an important preliminary step in data mining. Data cleaning algorithms attempt to smooth noise

☆ 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 [19].
* Corresponding author.
E-mail addresses: gjaganna@cs.stevens.edu (G. Jagannathan), rwright@cs.stevens.edu (R.N. Wright).

0169-023X/$ - see front matter
doi:10.1016/j.datak.2007.06.013

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 regulations often prevent the sharing of data between multiple parties. Privacy-preserving distributed data mining algorithms (e.g., [1,23]) allow cooperative computation of data mining algorithms without requiring the participating organizations to reveal their individual data items to each other. Existing privacy-preserving data mining algorithms—like most data mining algorithms—assume data is complete. In order to make use of those algorithms while maintain privacy in the data mining process, privacy-preserving methods of data cleaning are also required.

In this paper, we consider privacy-preserving imputation for data that is horizontally partitioned between two parties. Our main result is a privacy-preserving imputation algorithm based on ID3 decision trees. In the rest of this introduction, we place our results in the context of related work and describe our results in more detail.

1.1. Related work

Like data mining itself, pre-processing algorithms perform best when they have access to sufficient data. Pre-processing of distributed data has been previously investigated in the context of sensor networks [8,15]. However, these existing methods are driven by efficiency concerns and do not protect the privacy of the data. In our context, we view privacy as a primary concern (while also seeking as much efficiency as possible, as well as good imputation results).

Several methods for dealing with missing values have been proposed. One general approach to handling missing values is to create data mining algorithms that "internally" handle missing values and still produce good results. For example, the CART decision-tree learning algorithm [4] internally handles missing values essentially using an implicit form of imputation based on regression. However, in this paper, we follow the more common "modular" approach, where pre-processing is performed first and the resulting data is suitable for use with a variety of data mining algorithms. This is particularly needed in the setting of privacy-preserving data mining because, to date, the existing privacy-preserving data mining algorithms do not make any special internal handling of missing data. A stand-alone approach to privacy-preserving imputation can therefore be used in combination with any existing privacy-preserving data mining algorithm for the same distributed setting. In particular, our results in this paper are suitable for use with any privacy-preserving data mining algorithm for data that is horizontally partitioned between two parties (e.g., [23,20,18]).

Some of the simpler pre-processing techniques for handling missing data have limited applicability or introduce bias into the data [24]. One of the easiest approaches to dealing with missing values is simply to 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 [30]. 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 [12].

Data imputation replaces missing values with estimated values, typically producing 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 [6,16]) to more sophisticated ones that use regression [3], Bayesian networks [7], and decision-tree induction [21]. Using the mean or mode is generally considered a poor choice [24], 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 [11] 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 [22].

Classification is generally considered the best method for imputing missing data [10]. 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. A privacy-preserving linear regression protocol is presented in [9]; this would be useful for privacy-preserving 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 [28] or C4.5 [29] 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 [21,10]. It has been observed [21,10] that single imputation using decision trees is more accurate than imputation based on clustering. In some cases, the decision-tree construction can be *lazy* [12], in that only the needed path or paths of the tree is constructed. This has an efficiency advantage because it avoids time spent on constructing the parts of the tree that will not be needed.

Lindell and Pinkas [23] provide a privacy-preserving algorithm for computing the ID3 tree for databases that are horizontally partitioned between two parties. Although this is the same distributed setting we consider, we are unable to use their solution directly because it allows 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 (for efficiency reasons) we only construct the path that is needed and (for privacy reasons) we 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, which we describe in Section 2.3.

1.2. Our contributions

In this paper, we focus on decision-tree imputation because it generally produces superior results. Our main result is a privacy-preserving data imputation protocol for databases that are horizontally partitioned between two parties. Our solution uses a lazy decision-tree algorithm based on ID3 trees. As previously mentioned, the use of a lazy decision tree algorithm provides an efficiency improvement when only a small number of paths of the tree are needed. Our algorithm 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. The second version does not leak any information beyond the computed imputed value, but incurs slightly increased communication and computation costs.

We also briefly describe private protocols for data imputation based on other well-known imputation methods—namely, mean, mode, linear regression and clustering, noting that the simpler methods generally produce inferior results unless it is known that the data itself or the data mining methods to be applied on the processed data are suited to those methods.

We begin in Section 2 by introducing some definitions and existing subprotocols. We describe our privacy-preserving lazy decision tree imputation protocol in Section 3. In Section 4, we outline private protocols for data imputation using the mean, the mode, linear regression, and clustering-based prediction.

2. Preliminaries

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 subproblems for efficiency. In our distributed setting, there are two parties, who we call Alice and Bob. We use the notation $\alpha \in$ Alice to indicate that Alice holds the value or object α, and analogously, we write $\beta \in$ Bob if Bob holds β.

Alice and Bob have a horizontally partitioned database. That is, Alice holds a database $D_A = (d_1, \ldots, d_\ell) \in$ Alice and Bob holds a database $D_B = (d_{\ell+1}, \ldots, d_n) \in$ Bob, which are defined over a common set $\{A_1, \ldots, A_m\} \cup M$ of attributes. Because the attribute M will be the one in which there is a missing value, we also refer to it as the *class attribute*. Together, D_A and D_B form a single complete joint database $D = D_A \cup D_B$ with no missing values.[1] Additionally, there is another instance $I \in$ Bob (not included in $(d_1, \ldots, d_n)$) that 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. Ideally, nothing else about each other's data should be revealed to either party.

Throughout this paper, n denotes the number of instances in the joint database, m denotes the number of attributes, k denotes the maximum number of values any attribute can take, and g denotes the number of values the class attribute can take. Our solutions also make use of encryption in various places (usually indirectly through their subprotocols). We use t to denote the maximum number of bits required to represent any public key encryption and s to denote the maximum number of bits required to represent any symmetric key encryption.

In the rest of this section, we describe the lazy decision-tree algorithm on which our main distributed protocol is based, our privacy model, and some cryptographic primitives we use in our solutions.

2.1. Lazy decision-tree algorithm

In using a decision tree to predict a single value in a single instance, the entire decision tree is not needed. Rather, only a single path is traversed, whose values are determined by the instance containing the missing value. For efficiency reasons, we therefore use a lazy decision tree. We base our distributed solution on a lazy decision-tree algorithm that is a straightforward simplification of ID3.

In this section, we describe this solution as it would proceed in a centralized setting with access to all the joint data. This algorithm lends itself to an efficient privacy-preserving distributed solution. Comparing our algorithm to the LazyDT lazy decision-tree algorithm of Friedman et al. [12], their algorithm is more complex, less efficient, and less easily amenable for conversion to a privacy-preserving protocol, but also slightly more accurate. (Experiments in [12] that indicate that LazyDT, on average, has 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 decision-tree 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, shown in Fig. 1, starts 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 is said to be *pure* if all the instances in it have the same class label. The algorithm repeats the process of choosing an attribute of high information gain, and then winnowing the training data to those instances that match the test data on the chosen attribute. This process is repeated until the set of remaining instances is pure or all attributes have been exhausted. The algorithm then predicts the class label of the test instance as the most frequently occurring class in the remaining set of instances.

The algorithm does not directly calculate the attribute with the highest information gain. Instead, it calculates the attribute that results in the current set of instance having the least amount of entropy. Denote the current set (or subset) of instances by D, the set of values taken by the class attribute by $\{c_1, \ldots, c_g\}$, the set of values taken by an attribute A as $\{a_1, \ldots, a_k\}$, the set of instances of D in which A has value a_j by $D(a_j)$, and the number of instances of $D(a_j)$ in which the class label is c_i for $1 \leqslant i \leqslant g$ by p_{ji}. Then the conditional entropy after splitting on attribute A is defined as

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

[1] Technically, these databases are a sequences, not sets, but we occasionally abuse notation slightly and treat them as if they were sets.

Algorithm Lazy Decision Tree

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

Output: A label for instance I.

(1) If R is empty, return the majority label of instances in D.

(2) If D is pure, return the single occurring label c.

(3) Otherwise,

(a) for $i = 1$ to m
$\mathsf{Entropy}(A_i) = \mathsf{Entropy}(D, A_i)$

(b) $A_{\min}$ = Attribute with least entropy.

(c) $D = \{X \in D \mid X(A_{\min}) = I(A_{\min})\}$

(d) $R = R - \{A_{\min}\}$

(4) Go to Step 1

Fig. 1. Lazy decision tree algorithm.

The computation of this algorithm in a distributed privacy-preserving manner is the main result of this paper and is described in Section 3.

2.2. Privacy definition

The paradigm of secure distributed computation provides cryptographic solutions for protecting privacy in any distributed computation [31]. In that setting, the notion of privacy is defined by comparing the information that parties learn in carrying out a distributed protocol to the information that parties would learn if a trusted third party (TTP) were available to perform the computation for them. We use the same privacy definitions, but rather than using the general solutions provided by Yao for secure two-party computation [31], we provide a less general but more efficient solution for our specific two-party computation. We do, however, use general two-party computation as a building block for some smaller parts of our computation to design a tailored, more efficient, solution to privacy-preserving imputation.

As mentioned, our notion of privacy is in relation to the TTP setting in which there is a trusted third party to whom Alice and Bob send their data. The TTP 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 TTP setting. In this paper, we assume that both Alice and Bob are *semi-honest*. That is, both parties faithfully follow their specified protocols, but they may record intermediate messages in an attempt to infer information about the other party's data. The desired privacy condition in this model is that anything Alice or Bob learns from participating in the protocol could have been learned by simply giving them each their initial input and final output.

The privacy of our solution relies on composition of privacy-preserving protocols. In our solution, we use composition of a number of privacy-preserving subprotocols in which all intermediate outputs from one subprotocol that are inputs to the next subprotocol are computed as secret shares (see Section 2.3) Using composition, it follows that if each subprotocol produces only secret shares and is privacy-preserving, then the resulting composition is also privacy-preserving [14]. (A fully fleshed out proof of these results would require demonstrating algorithms called *simulators* showing that each party can simulate the interaction from their input and output alone.)

We note that there are advantages and limitations to this privacy definition. The definitions are quite strong, in that they state that nothing is learned by the parties in the distributed computation that they

would not learn if using a trusted third party. In particular, not only do parties not learn each other's data values unless they are implied by their outputs, they do not even gain a (non-negligible) advantage in guessing each other's data values correctly. However, these definitions do not address what the output itself reveals to the parties. It may very well be that in some cases, having parties learn imputed values based on their joint data might be considered too privacy-invasive even if they do not learn anything else about each other's data.

2.3. Cryptographic primitives

We use several existing cryptographic primitives, which we briefly describe in this section. We assume suitable hardness assumptions under which the various cryptographic primitives described in this section are secure.

2.3.1. Random shares

We say that *Alice and Bob have random shares* (or *secret shares*) of a value x to mean that x is divided into two pieces (*shares*) a and b such that Alice knows a, Bob knows b, x can be recovered from a and b, and x cannot be recovered without both a and b. We use both additive sharings and XOR sharings. In additive sharing, we choose N to be a large prime, where the field F is isomorphic to $\mathbb{Z}_N$. (In particular, N should be at least n.) Except where otherwise specified, all computations throughout the remainder of the paper take place in F. Alice and Bob have additive random shares of a value $x \in F$ if Alice knows a random value $a \in F$ and Bob knows a random value $b \in F$ such that $(a+b) \bmod N = x$. By XOR sharing, a bit x is shared as $x = a \oplus b$ for random bits a and b. An important property of additive shares is that they allow local computation of additions and subtractions in F. That is, if Alice and Bob share values x and y via random shares $a_x, a_y \in$ Alice and $b_x, b_y \in$ Bob, then $(a_x + a_y) \bmod N$ and $(b_x + b_y) \bmod N$ are random shares of $(x+y) \bmod N$. If $x + y \leqslant N$, then these are random shares of $x + y$.

2.3.2. Oblivious transfer

1-out-of-n oblivious transfer (denoted 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 and the sender learns nothing. Efficient solutions are given in [26,27]. When multiple OT operations are to be performed on the same data in sequence, better performance results can be obtained by using a protocol that carries them out all at once [17].

2.3.3. Yao's secure circuit-evaluation protocol

Yao's two-party secure circuit-evaluation protocol [31] allows two parties holding inputs a and b to privately compute any function $f(a, b)$ without revealing a and b to each other. The performance of Yao's protocol heavily depends on the size of a Boolean circuit for f and on the performance of OT_1^2. In theory, Yao's protocol could be applied to any distributed two-party privacy-preserving data mining problem. However, as a practical matter, the circuits for even simple computations on megabyte-sized databases would be intractably large. We make use of Yao's protocol for private computation in several cases, but only on functions involving a small number of small inputs.

2.3.4. Purity checking protocol

In a purity checking protocol, Alice has a vector of values $X = (x_1, \ldots, x_n)$ and Bob has a vector of values $Y = (y_1, \ldots, y_n)$. The protocol outputs c if $x_1 = \cdots = x_n = y_1 = \cdots = y_n = c$ (i.e., if the set of values in $X \cup Y$ is pure and includes only the value c), or $\perp$ otherwise. The parties learn nothing else. A simple purity checking protocol based on secure equality testing is described in [23].

2.3.5. Secure $x \ln(x)$ protocol

When Alice and Bob have random shares of x, denoted as v_1 and v_2, respectively, a secure $x \ln(x)$ protocol (such as the one in [23]), computes the shares of $x \ln(x)$ as w_1 and w_2 for Alice and Bob, respectively. The parties learn nothing else.

2.3.6. Private indirect index protocol

In a private indirect index protocol (PIX), Bob has a vector of values $X = (x_1, \ldots, x_n)$ and Alice and Bob have random XOR shares of an index i. That is, $i_1 \in$ Alice and $i_2 \in$ Bob and $i = i_1 \oplus i_2$. As the output of the protocol, Alice and Bob learn random shares of x_i. The parties learn nothing else. A PIX protocol is given in [25]. However, it outputs an XOR-sharing of x_i, while for our purposes we want an additive sharing instead. Fortunately, it can easily be modified to give an additive sharing instead. This protocol requires one invocation of OT_1^n.

2.3.7. Secure scalar product protocol

In a secure scalar product protocol (SPP), Alice has a vector $X = (x_1, \ldots, x_n)$ and Bob has a vector $Y = (y_1, \ldots, y_n)$. The protocol privately computes the scalar product as $X \cdot Y \equiv s_A + s_B \bmod N$, where s_A and s_B are random shares learned as output by Alice and Bob, respectively, and the parties learn nothing else. An SPP protocol is given in [13].

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. Recall the definitions of D_A, D_B and I given in Section 2 and recall that the class attribute M takes the values $\{c_1, \ldots, c_g\}$. 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.) In particular, Alice and Bob should learn nothing about the path of the decision tree that leads to the missing value including the remaining instances at each node and the value taken by the attributes along the path.

In Section 3.1, we first describe a basic version of the protocol. Besides the subprotocols already described in Section 2.3, the protocol requires four additional private subprotocols: a protocol that computes split entropy (shown in Section 3.3), a split protocol (shown in Section 3.4), a protocol that checks if the split database is pure (shown in Section 3.5), and a majority computation protocol (shown in Section 3.6). These subprotocols themselves use two additional subprotocols presented first in Section 3.2. We discuss the efficiency and privacy of the protocol in Section 3.7. As described, the basic version of our protocol has a small privacy leak. Specifically, it reveals to the parties the number of nodes traversed by the protocol in the undisclosed decision tree used for imputation. In Section 3.8, we show how 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 2.1 so that it functions more clearly as a data imputation algorithm. The lazy decision-tree algorithm computes as the root of the tree the attribute with the highest information gain. This is followed by the repeated execution of two steps until the remaining instances are pure or all attributes have been exhausted: First, extract the subset of the instances that match I on the chosen attribute. Second, choose an attribute of high information gain for the next iteration. Once the repetition of those two steps has completed, the algorithm outputs to Bob the missing value the majority label on the remaining instances. (Again, if Alice started with the missing value, their roles would be reversed.)

Our privacy-preserving protocol follows the same basic outline. However, the 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. To meet these requirements, we make extensive use of random sharings. 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 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 $p_i \oplus q_i = 1$ if $d_i \in D'$ and $p_i \oplus q_i = 0$ otherwise, for $1 \leqslant i \leqslant n$. The entropy computed for each attribute in each iteration is also held as random shares by the two parties.

We now describe our protocol in more detail. The complete protocol is shown in Fig. 2. At the beginning of the protocol, Alice and Bob jointly check if the database D is pure. If D is pure, Bob learns the unique label. This is done without revealing either their data or one-sided information about purity to each other using a purity checking protocol (see Section 2.3).

To compute the root of the lazy decision tree, Alice and Bob compute random shares of the entropy for every attribute $\{A_1, \ldots, A_m\}$. At the root level, computing the shares of the entropy is straightforward. The conditional entropy of a database D (as explained in Section 2.1) with respect to the attribute A can be rewritten as

Protocol Private Lazy Decision Tree Data Imputation

Input: A database $D = D_A \cup D_B$ of labeled instances (with attributes $\{A_1, \ldots, A_m\} \cup M$), 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 learns $I(M)$.

(1) Using a purity checking protocol (Section 2.3), if D is pure, Bob learns unique 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, using a secure $x \log x$ protocol (Section 2.3) and local computation, 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 (Section 2.3), Alice and Bob compute $\min_A$ and $\min_B$ such that $\min_A \oplus \min_B = \min$ where $\mathsf{Ent}_{\min}^A + \mathsf{Ent}_{\min}^B \equiv \text{minimum}\{\mathsf{Ent}_i^A + \mathsf{Ent}_i^B\}, 1 \leq i \leq m$.

(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 PIX (Section 2.3) with inputs $\min_A \in$ Alice and I, $\min_B \in$ Bob.
      - Alice and Bob run the Secure Split protocol of Section 3.4 on each of their instances 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 $p_i \oplus q_i = 0$ otherwise.
   (b) Using the Subset Purity Checking protocol of Section 3.5, if D_j is pure, Bob learns the unique label and exits. Otherwise:
      - for $i = 1$ to m, Alice and Bob run the Secure Split Entropy protocol of Section 3.3 on D_j, P, Q, and A_i to learn shares Ent_A^i and Ent_B^i.
      - Using Yao's protocol (Section 2.3), Alice and Bob compute $\min_A$ and $\min_B$ such that $\min_A \oplus \min_B = \min$ where $\mathsf{Ent}_{\min}^A + \mathsf{Ent}_{\min}^B \equiv \text{minimum}\{\mathsf{Ent}_i^A + \mathsf{Ent}_i^B\}, 1 \leq i \leq m$.

(4) Using the Majority Label protocol of Section 3.6, Bob learns the majority label to use as the missing value $I(M)$.

Fig. 2. Private lazy decision tree imputation protocol.

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

where $p_{j\ell}$ and $D(a_j)$ are as explained in Section 2.1. Because the database is horizontally partitioned between Alice and Bob, they can compute shares of $p_{j\ell}$ and $D(a_j)$ independently. Using a secure $x\log x$ protocol (see Section 2.3) and local additions and subtractions, Alice and Bob can jointly compute a random sharing of each $\mathsf{Entropy}(D,A_i)$.

After this computation, Alice has a vector $(\mathsf{Ent}_1^A,\ldots,\mathsf{Ent}_m^A)$ of entropy shares and Bob, correspondingly, has $(\mathsf{Ent}_1^B,\ldots,\mathsf{Ent}_m^B)$ such that $\mathsf{Ent}_i^A + \mathsf{Ent}_i^B \equiv \mathsf{Entropy}(D,A_i) \bmod N$ for $1 \leqslant i \leqslant m$. The index of the attribute 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 $\mathsf{Ent}_i^A + \mathsf{Ent}_i^B$ 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 requires that the attributes that have been chosen at various stages in the path of the lazy decision-tree computation not be revealed to either party. To achieve this, 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 PIX. 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 instance ($1 \leqslant i \leqslant n$) such that

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

The protocol for performing this computation privately is described in Section 3.4. 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 learns the unique label using the secure protocol that checks if D_j is pure and returns the unique label. It is important to observe that since neither party knows which of their instances are in D_j, we cannot use here kind of the purity checking protocol described in Section 2.3. Instead, we provide an alternate variation of a purity checking protocol, described in Section 3.5, that works on a secret-shared representation of the database as required in our case. If D_j is not pure, Alice and Bob engage in the secure entropy computation protocol of Section 3.3 for the split D_j for each of the attributes $\{A_1,\ldots,A_m\}$, and compute the random shares of the split entropy $\mathsf{Entropy}(D,A_i)$. (i.e., $\mathsf{Ent}_i^A + \mathsf{Ent}_i^B \equiv \mathsf{Entropy}(D,A_i) \bmod N$) The attribute with the least entropy is computed using 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 is zero for any attribute that has already been chosen at a previous level.

3.2. Secure computation of the number of instances

In this section, we present two protocols for computing shares of the total number of instances in a subset of the database that is only identifiable by random shares held by Alice and Bob, respectively.

3.2.1. Secure Hamming distance

A secure Hamming distance protocol, shown in Fig. 3, is useful for us when applied to a subset $D' \subseteq D$ represented by bit vectors $X = (x_1,\ldots,x_n)$ and $Y = (y_1,\ldots,y_n)$ known to Alice and Bob, respectively, where $x_i \oplus y_i = 1$ if instance $d_i \in D'$ and $x_i \oplus y_i = 0$ otherwise. At the end of the protocol, Alice and Bob learn random shares of this Hamming distance, which is also equal to $|D'|$. The protocol uses one invocation of SPP. Because of the linear relationship for binary data between the scalar product and the Hamming distance, the parties are then able to compute their needed results locally.

Protocol Hamming Distance

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

Output: Alice and Bob learn α and β, respectively, such that $(\alpha + \beta) \bmod N = v$, where v is the Hamming distance of X and Y—i.e., total number of entries for which $x_i \oplus y_i = 1$.

(1) Alice and Bob run SPP on X and Y to securely compute shares $\mu \in$ Alice and $\lambda \in$ Bob of their scalar product.

(2) Alice computes $\sum_{i=1}^{n} x_i = \gamma$.

(3) Bob computes $\sum_{i=1}^{n} y_i = \delta$.

(4) Alice computes $\alpha = \gamma - 2\mu$ and Bob computes $\beta = \delta - 2\lambda$.

Fig. 3. Secure Hamming distance.

The only communication that happens between the two parties is during the invocation of SPP using two vectors of size n. The communication complexity of the SPP protocol of [13] is $O(tn)$. (Recall that t denotes the number of bits needed to represented a public key encryption.) Its computation complexity involves n encryptions and one decryption for Alice and n modular exponentiations and one encryption for Bob. The SPP protocol is secure and it provides no useful information other than the random shares to the two parties. Hence, this protocol is secure.

3.2.2. Secure computation of the total number of instances in a subset with a given class label

Let D'_c denote the set of instances in D' in which the attribute M takes the value c. This protocol takes as input a subset $D' \subseteq D$ and a value $c \in \{c_1, \ldots, c_g\}$ and computes shares of total number of instances in the set D'_c. The input 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 $x_i \oplus y_i = 1$ if instance $d_i \in D'$ and $x_i \oplus y_i = 0$ otherwise.

Alice and Bob first locally determine which of their instances have the right class value c. Specifically, for $1 \leqslant i \leqslant \ell$, Alice computes $w_i = 1$ if $d_i(M) = c$ and $w_i = 0$ otherwise; Bob does the same for $\ell + 1 \leqslant i \leqslant n$. Then the total number of instances in D' with class label c is the number of the instances that satisfy the relation $(x_i \oplus y_i) \wedge w_i = 1$ for $1 \leqslant i \leqslant n$. This relation can be rewritten as $(\overline{x_i} w_i) y_i \vee (x_i w_i) \overline{y_i} = 1$. Since both $(\overline{x_i} w_i) y_i$ and $(x_i w_i) \overline{y_i}$ cannot hold at the same time, this number is equal to the sum of the number of instances satisfying $(\overline{x_i} w_i) y_i = 1$ and the number of instances satisfying $(x_i w_i) \overline{y_i} = 1$. This is computed as random shares between Alice and Bob using SPP. The complete protocol is shown in Fig. 4.

The communication and computation complexity are the same as that of SPP. The privacy of this protocol follows from the privacy of the SPP protocol.

3.3. 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. The subset D' is represented by two bit vectors P and Q known to Alice and Bob, respectively, such that $p_i \oplus q_i = 1$ if $d_i \in D'$ and $p_i \oplus q_i = 0$ otherwise, for $1 \leqslant i \leqslant n$.

The protocol computes random shares of the entropy after splitting D' on the attribute A, which takes on values $a_1, \ldots, a_k$. In the decision-tree computation, the entropy after splitting the database D'on attribute A is as shown in Eq. (1) in Section 3.1. Alice and Bob use Yao's protocol to privately compute an XOR sharing representation of $D(a_j) \subset D'$ for $1 \leqslant j \leqslant k$. That is, $D(a_j)$ is represented by two bit vectors $R = (R_1, \ldots, R_n) \in$ Alice and $S = (S_1, \ldots, S_n) \in$ Bob, where for $1 \leqslant i \leqslant n, R_i \oplus S_i = 1$ if $p_i \oplus q_i = 1$ and $d_i(A) = a_j$; $R_i \oplus S_i = 0$ otherwise. Using the protocols of Section 3.2, Alice and Bob compute random shares of $|D(a_j)|$ and $p_{j\ell}$ for

Protocol Total Instances with Class Label

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

Output: Alice and Bob learn γ and δ, respectively, such that $\gamma + \delta \equiv |D'_c| \bmod N$, where $|D'_c|$ is the total number of instances for which $x_i \oplus y_i = 1$ and M takes the value c.

(1) Alice computes a vector $(w_1, \ldots, w_\ell)$ with $w_i = 1$ if $d_i(M) = c$ and $w_i = 0$ otherwise, for $1 \leq i \leq \ell$.

(2) Bob computes a vector $(w_{\ell+1}, \ldots, w_n)$ with $w_i = 1$ if $d_i(M) = c$ and $w_i = 0$ otherwise, for $\ell + 1 \leq i \leq n$.

(3) Alice and Bob run SPP four times, as follows:

- inputs $(\overline{x_1} w_1, \ldots, \overline{x_\ell} w_\ell)$ and $(y_1, \ldots, y_\ell)$; outputs α_1 and β_1.
- inputs $(x_1 w_1, \ldots, x_\ell w_\ell)$ and $(\overline{y_1}, \ldots, \overline{y_\ell})$; outputs α_2 and β_2.
- inputs $(\overline{x_{\ell+1}}, \ldots, \overline{x_n})$ and $(w_{\ell+1} y_{\ell+1}, \ldots, w_n y_n)$; outputs γ_1 and δ_1.
- inputs $(x_{\ell+1}, \ldots, x_n)$ and $(w_{\ell+1} \overline{y_{\ell+1}}, \ldots, w_n \overline{y_n})$; outputs γ_2 and δ_2.

(4) Alice computes $\gamma = \alpha_1 + \alpha_2 + \gamma_1 + \gamma_2$ and Bob computes $\delta = \beta_1 + \beta_2 + \delta_1 + \delta_2$.

Fig. 4. Secure computation of $|D'_c|$.

$1 \leqslant \ell \leqslant g$ and $1 \leqslant j \leqslant k$. Random shares of the terms $p_{j\ell} \log p_{j\ell}$, for $1 \leqslant \ell \leqslant g$, are computed using a secure $x \log x$ protocol. The complete Secure Split Entropy protocol is shown in Fig. 5.

The size of the circuit in Step 1 is $O(\log N)$ and this circuit evaluation occurs n times. Hence, the communication complexity is $O(ns \log N)$. The computation complexity is $O(n \log N)$ invocations of OT_1^2. Step 1b involves two parties jointly computing random shares of $|D(a_j)|$ using the protocol described in Section 3.2.1. Step 1c uses g invocations of the Total Instances with Class Label protocol of Section 3.2.2. The $x \log x$ protocol of [23] has a communication complexity of $O(s \log N)$ bits and a computation complexity equal to that of $O(\log N)$ OT_1^2 invocations.

Thus, the total communication complexity of the Secure Split Entropy protocol is $O(tkng + (g + n)ks \log N)$. The total computation complexity is the cost of $O((n + g)k \log N)$ OT_1^2 invocations plus the cost of $O(gk)$ SPP invocations, which in turn involves a total of $O(gnk)$ encryptions and $O(gk)$ decryptions for Alice and $O(ngk)$ modular exponentiations and $O(gk)$ encryptions for Bob.

The output of the Secure Split Entropy protocol is a random sharing of the entropy after splitting D' on the attribute A. Yao's protocol securely computes the subset $D(a_j)$ of the database whose instances take a specific value for the attribute A. This subset is represented as a random sharing between the two parties. The subprotocols in Section 3.2 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 $x \log x$ protocol is also secure. Thus, the Secure Split Entropy protocol securely computes random shares of the entropy after splitting D' on the attribute A.

3.4. Secure split protocol

The Secure Split protocol is used to determine if a given instance is in a subset $D' \subseteq D$ and attribute A_i takes the value v in that instance. The result of this test (0 or 1) is randomly shared between Alice and Bob. Here, inputs v and i are shared between Alice and Bob as $v \equiv a + b \bmod N$ 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 instance in D' is represented by two bits $p \in$ Alice and $q \in$ Bob such that $p \oplus q = 1$ if the instance is in D' and $p \oplus q = 0$ otherwise.

Protocol Secure Split Entropy

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 (which takes on values $a_1, \ldots, a_k$), and $D' \subseteq D$ represented by 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 $d_i \in D'$, and $p_i \oplus q_i = 0$ otherwise, for $1 \leq i \leq n$.

Output: Random shares of the entropy resulting from splitting D' on attribute A.

(1) 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 $d_i(A)$.

(b) Alice and Bob engage in the Hamming Distance protocol of Section 3.2.1 on R and S to compute random shares of $|D(a_j)|$.

(c) To compute shares of the number $p_{j\ell}$ of instances of $D(a_j)$ in which the class label is c_ℓ for $1 \leq \ell \leq g$, Alice and Bob engage in g executions of the Total Instances with Class Label protocol of Section 3.2.2.

(d) They use a secure $x \log x$ protocol to compute random shares of $|D(a_j)| \log |D(a_j)|$, $p_{j\ell} \log p_{j\ell}$ for $1 \leq \ell \leq g$.

(e) They locally compute random shares of $\mathsf{sum}_j = |D(a_j)| \log |D(a_j)| - \sum_{\ell=1}^{g} (p_{j\ell} \log p_{j\ell})$.

(2) Alice and Bob compute random shares of entropy for the attribute A by locally adding their own shares obtained in Step 1e.

Fig. 5. Secure split entropy.

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

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

In other words, the fact that a given attribute has a specific value in a given instance is available to both Alice and Bob only as random shares.

This protocol has two stages. In the first stage, Alice and Bob use PIX to learn results x and y, respectively, where $(x + y) \equiv v_i \bmod N$. In the second stage, Alice has inputs x, a, and p, and Bob has inputs y, b, and q. They use Yao's protocol to learn two bits $b_1 \in$ Alice and $b_2 \in$ Bob such that

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

The communication and computation complexity of the first stage is that of one invocation of OT_1^m. The second stage of the protocol involves one comparison which is done using Yao's protocol, requiring $O(s \log N)$ bits of communication and $O(\log N)$ invocations of OT_1^2.

The first stage of the protocol uses PIX, which is secure and produces only random shares as its results. Yao's protocol is also secure and does not leak any information. Thus, the Secure Split protocol is secure.

3.5. Secure protocol to check if D is pure

The Subset Purity Checking protocol takes as input a subset D' of a database D and outputs to Bob the value c if all the instances in D' have the same label c or $\perp$ otherwise. In contrast to the purity checking primitive of Section 2.3, in this case, Alice and Bob only have an XOR sharing of D'—that is, 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 $p_i \oplus q_i = 1$ if $d_i \in D'$ and $p_i \oplus q_i = 0$ otherwise.

Let $|D'_c|$ denote the number of instances in D' where the class attribute takes value c. For $1 \leqslant i \leqslant g$, Alice and Bob compute random shares α_i^A and α_i^B of $|D'_{c_i}|$ using the Total Instances with Class Label protocol of Section 3.2.2. Alice and Bob then use Yao's protocol to check whether there exists j such that $(\alpha_j^A + \alpha_j^B) \neq 0 \bmod N$ and $(\alpha_i^A + \alpha_i^B) \equiv 0 \bmod N$ for every $i \neq j$. If so, Bob's output from Yao's protocol is c_j; otherwise, it is $\perp$.

Yao's protocol for the comparison described above requires $O(sg \log N)$ bits of communication and $O(g \log N)$ invocations of OT_1^2. The communication and the computation complexity is dominated by the g executions of the Total Instances with Class Label protocol. The Total Instances with Class Label protocol returns only random shares to Alice and Bob. Yao's protocol is secure and does not leak any information. Hence, the Subset Purity Checking protocol is secure, in that it does not leak anything other than its output. We note however that this protocol breaks our convention that its output is secret shares. We discuss the resulting privacy implications in Section 3.7.

3.6. Secure protocol for majority label

The Majority Label protocol is similar to the Subset Purity Checking protocol of Section 3.5, except that it always returns the majority class label even if its input database is not pure. In the first stage, for $1 \leqslant i \leqslant g$, Alice and Bob use the Total Instances with Class Label protocol of Section 3.2.2 to compute random shares $\alpha_i^A \in$ Alice and $\alpha_i^B \in$ Bob of $|D'_{c_i}|$. In the second stage, Alice and Bob use Yao's protocol with inputs α_i^A and α_i^B, respectively, for Bob to learn the index $j = \operatorname{argmax}_{1 \leqslant i \leqslant g} |D'_{c_i}|$. Bob computes c_j as the majority label. The complexity of this protocol is the same as the purity protocol. It reveals nothing beyond the majority label c_j to Bob and nothing to Alice.

3.7. Performance and privacy analysis

Having defined all the subprotocols required by the Private Lazy Decision Tree Data Imputation protocol, we now return to its complexity and privacy.

3.7.1. Complexity

Alice and Bob communicate with each other to compute the attribute with the maximum information gain at each level of the path. Putting together the cost of all the involved subprotocols, we see that the communication complexity is dominated by $O(m^2 nk(tg + s \log N))$. The total computation complexity is that of $O(m^2 k(n+g) \log N)$ OT_1^2 invocations, $O(m)$ OT_1^m invocations, and $O(m^2 kg)$ SPP invocations. The latter involves $O(m^2 kgn)$ encryptions and $O(m^2 kg)$ decryptions for Alice and $O(m^2 kgn)$ modular exponentiations and $O(m^2 kg)$ encryptions for Bob.

Our solution offers at least a log-factor improvement over using Yao's generic solution. Using Yao's protocol for the entire computation would require starting with a Boolean circuit computing the result of the ID3 decision tree. The Boolean circuit size for this function will be at least $\Omega(m^2 kn \log(n) \log N)$. The resulting private computation requires a number of invocations of OT_1^2 that is equal to the input size in bits—i.e., $nm \log(N)$. The resulting communication complexity is $\Omega(m^2 kn \log(n) \log N)$ because of the circuit size. In comparison, our solution, including the invocations of Yao's protocol where we require it, is dominated by $O(m^2 nk)$ circuits of input size $O(\log N)$. The number of invocations of OT_1^2 is $O(m^2 nk \log N)$ and the communication complexity is $O(m^2 nk \log N)$, for a total savings of a factor of $\log(n)$ in communication.

While this is likely not be efficient enough for very large data sets or in every setting, when data privacy is sufficiently important to the parties involved and they want to use their joint data, it may be preferable to incur this performance penalty than to avoid engaging in the computation at all.

3.7.2. 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 desired imputed value.

In the Private Lazy Decision Tree Data 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. We have already shown that all the intermediate outputs of the subprotocols are held only as random shares by Alice and Bob and that all the subprotocols do not leak any information beyond Bob learning the final imputation result, with one exception. This exception is that the Subset Purity Checking protocol reveals to Bob (and indirectly to Alice, by whether the computation then terminates or continues) whether the split database at each node is pure or not. Because the algorithm gradually splits the database until either the database is pure or all the attributes are exhausted, knowing this information is equivalent to knowing the length of the path. That is, in executing the Private Lazy Decision Tree Data Imputation protocol, Bob learns the imputed value, but also both Alice and Bob learn the number of attributes in the path in the decision tree (though not which attributes, nor the values they take), which would not be learned in the TTP solution. The composition result (see Section 2.2) implies that they do not learn anything else; that is, Private Lazy Decision Tree Data Imputation can be viewed as a private protocol computing an output to Alice consisting of the length of the path in the decision tree and to Bob consisting of the length of the path and the learned value for the missing attribute.

In Section 3.8, we describe a modification of the protocol that is completely private, not revealing to Alice and Bob the length of the path.

3.8. Enhancing privacy

As just discussed, the Private Lazy Decision Tree Data Imputation protocol 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 modification that uses additional secret sharing to eliminate this leak, thus achieving the same level of privacy as in the TTP model.

The modified protocol uses two additional variables: is_pure and majority_class. We treat is_pure as Boolean (with $0 = \mathsf{false}$ and $1 = \mathsf{true}$). Each of these variables is randomly shared between Alice and Bob, via shares $\mathsf{is\_pure}^A$, $\mathsf{is\_pure}^B$, $\mathsf{majority\_class}^A$, and $\mathsf{majority\_class}^B$. At the end of each iteration, the protocol ensures that $\mathsf{is\_pure}^A \oplus \mathsf{is\_pure}^B = \mathsf{is\_pure}$ and $\mathsf{majority\_class} \equiv (\mathsf{majority\_class}^A + \mathsf{majority\_class}^B) \bmod N$. Initially, is_pure is set to false by setting $\mathsf{is\_pure}^A = \mathsf{false}$ and $\mathsf{is\_pure}^B = \mathsf{false}$. If is_pure becomes true at the end of some iteration, it means that the protocol has determined that the database 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 database D.

The modified protocol requires changes to Subset Purity Checking and Majority Label (Sections 3.5 and 3.6). In the modified version of Subset Purity Checking, $\mathsf{is\_pure}^A$, $\mathsf{is\_pure}^B$, $\mathsf{majority\_class}^A$, and $\mathsf{majority\_class}^B$ are supplied as additional inputs and the protocol checks if is_pure is false. If so, the protocol computes random shares of the new values of is_pure and majority_class and returns these to Alice and Bob. On the other hand, if is_pure is true when the subprotocol is invoked, then the values of is_pure and majority_class do not change, but a new random sharing of each is computed and sent to Alice and Bob.

In the modified version of Majority Label, the values of $\mathsf{is\_pure}^A$, $\mathsf{is\_pure}^B$, $\mathsf{majority\_class}^A$ and $\mathsf{majority\_class}^B$ are supplied as additional inputs. 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 $\mathsf{majority\_class}^A$ and $\mathsf{majority\_class}^B$. On the other hand, if is_pure is true, the computation of the majority label is skipped. Instead, $(\mathsf{majority\_class}^A + \mathsf{majority\_class}^B) \bmod N$ is used in place of majority_class, and this is again randomly shared as $\mathsf{majority\_class}^A$ and $\mathsf{majority\_class}^B$.

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 if a pure set D is obtained before all attributes have been used. (Indeed, it could not, as now neither participant knows that this has occurred.) At the end of all iterations, Alice sends the value of $\mathsf{majority\_class}^A$ to Bob who then uses $\mathsf{majority\_class}^A + \mathsf{majority\_class}^B \bmod N$ as the imputed value. This eliminates the earlier privacy leak and results in a private protocol for Bob to learn the desired imputed value.

4. 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. Let $\alpha_1, \ldots, \alpha_n$ denote the values of the attribute M for which $I \in$ Bob has a missing value for the instances $d_1, \ldots, d_n$.

4.1. Mean

Because Bob will inevitably be able to compute the sum of Alice's values for attribute M from the mean of their combined values, we can satisfy the privacy requirements described in Section 2.2 with a very simple protocol. 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 computes the missing value $I(M)$ as $(a+b)/n$.

4.2. Mode

Alice computes the frequencies of the possible values $(c_1, \ldots, c_g)$ of M in her database D_A as $\alpha_1, \ldots, \alpha_g$ and Bob computes the frequencies in his database D_B as $\beta_1, \ldots, \beta_g$. Alice and Bob then use Yao's protocol with inputs $\alpha_1, \ldots, \alpha_g$ and $\beta_1, \ldots, \beta_g$ for Bob to learn $j = \mathrm{argmax}_{1 \leqslant i \leqslant g}(\alpha_i + \beta_i)$. Bob then takes c_j as the missing value.

4.3. 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 $m = \frac{\sum_{i=i}^{n}(x_i y_i) - \left(\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i\right)/n}{\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2 / n}$ and $b = \frac{\sum_{i=1}^{n} x_i}{n} - m \cdot \frac{\sum_{i=1}^{n} y_i}{n}$. The operations involved are linear, so Alice and Bob can securely compute shares of m and b. Using these shares, Bob can compute the missing value.

4.4. Clustering

Alice and Bob use a secure clustering protocol (such as the one in [18]) to compute shares of m cluster centers, where m is a user-specified parameter. Let $C_i^A = (a_{i1}^A, \ldots, a_{ik}^A, p_i^A)$ and $C_i^B = (a_{i1}^B, \ldots, a_{ik}^B, p_i^B)$, for $1 \leqslant i \leqslant m$, denote the shares of the m cluster centers for Alice and Bob, respectively. Let the instance $I = (x_1, \ldots, x_k, x)$, 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 $Z^A = (z_1^A, \ldots, z_m^A)$ and $Z^B = (z_1^B, \ldots, z_m^B)$ denote Alice's and Bob's shares of the distances, respectively. They use Yao's protocol to compute random XOR shares i_1 and i_2 of the index i such that $z_i^A + z_i^B = \min_{1 \leqslant j \leqslant m}(z_j^A + z_j^B)$. Bob should learn the missing value as $p_i^A + p_i^B$, where p_i^A and p_i^B are Alice and Bob's shares, respectively, of the missing attribute in cluster C_i. To do this, Alice and Bob then invoke PIX twice, once for the vector $(p_1^A, \ldots, p_m^A)$ and the other for the vector $(p_1^B, \ldots, p_m^B)$. 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.

5. Conclusions

Privacy-preserving methods of pre-processing data 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 pre-processing, then privacy is lost, while if pre-processing is not performed on the joint data, then the data mining results are likely to be inaccurate.

In this paper, we provide a first step in this direction—namely, a privacy-preserving solution to data imputation. 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.

Future directions include private solutions for other data imputation methods, handling vertically partitioned data, extending our results to the multiparty case, and strengthening our results to withstand malicious parties.

References

[1] R. Agrawal, R. Srikant, Privacy-preserving data mining, in: ACM SIGMOD, May 2000, pp. 439–450.
[2] G.E.A.P.A. Batista, M.C. Monard, An analysis of four missing data treatment methods for supervised learning, Applied Artificial Intelligence 17 (5) (2003) 519–533.
[3] J.-F. Beaumont, On regression imputation in the presence of nonignorable nonresponse, in: Proceedings of the Survey Research Methods Section, ASA, 2000, pp. 580–585.
[4] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Chapman and Hall, 1984.
[5] J. Chen, J. Shao, Nearest neighborhood imputation for survey data, Journal of Official Statistics 16 (2) (2000) 113–131.
[6] P. Clark, T. Niblett, The CN2 induction algorithm, Machine Learning 3 (4) (1989) 261–283.
[7] L. Coppola, M. Di Zio, O. Luzi, A. Ponti, M. Scanu, Bayesian networks for imputation in official statistics: a case study, in: DataClean Conference, 2000, pp. 30–31.
[8] I. Davidson, S.S. Ravi, Distributed pre-processing of data on networks of berkeley motes using non-parametric EM, in: Proceedings of SIAM SDM Workshop on Data Mining in Sensor Networks, 2005, pp. 17–27.
[9] W. Du, Y. Han, S. Chen, Privacy-preserving multivariate statistical analysis: Linear regression and classification, in: Proceedings of the Fourth SIAM International Conference on Data Mining, 2004, pp. 222–233.
[10] A. Farhangfar, L. Kurgan, W. Pedrycz, Experimental analysis of methods for handling missing values in databases, in: Intelligent Computing: Theory and Applications II, 2004.
[11] B.L. Ford, Incomplete data in sample surveys, An Overview of Hot-deck Procedures, Academic Press, 1983.
[12] J.H. Friedman, R. Kohavi, Y. Yun, Lazy decision trees, in: Proceedings of 13th AAAI and 8th IAAI, 1996, pp. 717–724.
[13] B. Goethals, S. Laur, H. Lipmaa, T. Mielikainen, On secure scalar product computation for privacy-preserving data mining, in: 7th ICISC, 2004.
[14] O. Goldreich, Foundations of Cryptography, vol. II, Cambridge University Press, 2004.
[15] M. Halatchev and L. Gruenwald, Estimating missing values in related sensor data streams, in: Proceedings of the Eleventh International Conference on Management of Data, 2005, pp. 83–94.
[16] M. Hu, S.M. Salvucci, M.P. Cohen, Evaluation of some popular imputation algorithms, in: The Survey Research Methods Section of the ASA, 1998, pp. 308–313.
[17] Y. Ishai, J. Kilian, K. Nissim, E. Petrank, Extending oblivious transfer efficiently, in: Advances in Cryptology – CRYPTO '03: Proceedings of the 23rd Annual International Cryptology Conference, Lecture Notes in Computer Science, vol. 2729, Springer-Verlag, 2003, pp. 145–161.
[18] G. Jagannathan, K. Pillaipakkamnatt, R. Wright, A new privacy-preserving distributed k-clustering algorithm, in: Proceedings of the 6th SIAM International Conference on Data Mining, 2006, pp. 492–496.
[19] G. Jagannathan, R.N. Wright, Privacy-preserving data imputation, in: Proceedings of the ICDM International Workshop on Privacy Aspects of Data Mining, 2006, pp. 535–540.
[20] M. Kantarcioglu, J. Vaidya, Privacy-preserving naive Bayes classifier for horizontally partitioned data, in: IEEE Workshop on Privacy-Preserving Data Mining, 2003.
[21] K. Lakshminarayan, S.A. Harp, T. Samad, Imputation of missing data in industrial databases, Applied Intelligence 11 (3) (1999) 259–275.
[22] R.C.T. Lee, J.R. Slagle, C.T. Mong, Towards automatic auditing of records, IEEE Transactions on Software Engineering 4 (5) (1978) 441–448.
[23] Y. Lindell, B. Pinkas, Privacy preserving data mining, Journal of Cryptology 15 (3) (2002) 177–206.
[24] R.J.A. Little, D.B. Rubin, Statistical Analysis with Missing Data, John Wiley & Sons, Inc., USA, 1986.
[25] M. Naor, K. Nissim, Communication preserving protocols for secure function evaluation, in: STOC, 2001, pp. 590–599.
[26] M. Naor, B. Pinkas, Oblivious transfer and polynomial evaluation, in: 31st STOC, 1999, pp. 245–254.

[27] M. Naor, B. Pinkas, Efficient oblivious transfer protocols, in: 12th SODA, 2001, pp. 448–457.
[28] J.R. Quinlan, Induction of decision trees, Machine Learning 1 (1) (1986) 81–106.
[29] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
[30] W. Sarle, Prediction with missing inputs, in: Proceedings of the Fourth Joint Conference on Information Sciences, 1998, pp. 399–402.
[31] A.C.-C. Yao, How to generate and exchange secrets, in: 27th FOCS, 1986, pp. 162–167.

Geetha Jagannathan is a Ph.D. student in the Computer Science Department at Stevens Institute of Technology in Hoboken, New Jersey. Her research interests lie in cryptography, privacy and computational number theory. She received a Ph.D. in Mathematics from the Indian Institute of Technology, Chennai in 1994. She is a member of the IEEE, and the ACM.

Rebecca Wright is a Professor in the Computer Science Department at Stevens Institute of Technology in Hoboken, New Jersey. Her research spans the area of information security, including cryptography, privacy, foundations of computer security, and fault-tolerant distributed computing. She serves as an editor of the Journal of Computer Security (IOS Press) and the International Journal of Information and Computer Security (Inderscience), and was a member of the board of directors of the International Association for Cryptologic Research from 2001 to 2005. She was Program Chair of Financial Cryptography 2003 and the 2006 ACM Conference on Computer and Communications Security and was General Chair of Crypto 2002. She received a Ph.D. in Computer Science from Yale University in 1994 and a B.A. from Columbia University in 1988. She is a member of the IEEE, the ACM, and the IACR.