Tools for Privacy Preserving Distributed Data Mining

Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya
Purdue University
Department of Computer Sciences
250 N University St
West Lafayette, IN 47907-2066 USA
(clifton, kanmurat, jsvaidya)@cs.purdue.edu

Xiaodong Lin, Michael Y. Zhu
Purdue University
Department of Statistics
150 N University St
West Lafayette, IN 47907-2067 USA
(linxd, yuzhu)@stat.purdue.edu

ABSTRACT

Privacy preserving mining of distributed data has numerous applications. Each application poses different constraints: What is meant by privacy, what are the desired results, how is the data distributed, what are the constraints on collaboration and cooperative computing, etc. We suggest that the solution to this is a *toolkit* of components that can be combined for specific privacy-preserving data mining applications. This paper presents some components of such a toolkit, and shows how they can be used to solve several privacy-preserving data mining problems.

Keywords

Privacy, Security

1. INTRODUCTION

Data mining has operated on a data warehousing model of gathering all data into a central site, then running an algorithm against that data. Privacy considerations may prevent this approach. For example, the Centers for Disease Control may want to use data mining to identify trends and patterns in disease outbreaks, such as understanding and predicting the progression of a flu epidemic. Insurance companies have considerable data that would be useful – but are unwilling to disclose this due to patient privacy concerns. An alternative is to have each of the insurance companies provide some sort of statistics on their data that cannot be traced to individual patients, but can be used to identify the trends and patterns of interest to the CDC.

Privacy-preserving data mining has emerged to address this issue, with several papers in the past few years as well as articles in the popular press[7; 10]. One approach is to alter the data before delivering it to the data miner, this is discussed elsewhere in this issue[8]. The second approach assumes the data is distributed between two or more sites, and these sites cooperate to learn the global data mining results without revealing the data at their individual sites. This approach was first introduced to the data mining community by Lindell and Pinkas[13], with a method that enabled two parties to build a decision tree without either party learning *anything* about the other party's data, except what might be revealed through the final decision tree. We have since developed techniques for association rules[12; 17], clustering, k-nearest neighbor classification, and are working on others.

In our research on this subject, we have come to two observations that we feel should guide further work:

1. For each data mining approach, there are many *interesting* privacy-preserving distributed data mining problems. For example, Naïve Bayes is a classic approach to classification. However, the way the data is partitioned between parties, varied privacy constraints, and communication/computation considerations lead to *several* privacy-preserving solutions to Naïve Bayes.
2. While there may be many different data mining techniques, they often perform similar computations at various stages (e.g., counting the number of items in a subset of the data shows up in both association rule mining and learning decision trees.)

From observation 1, we feel the current approach of developing and publishing solutions to individual privacy-preserving data mining problems will generate more papers than real-world solutions. While such research is necessary to understand the problem, a myriad of solutions is difficult to transfer to industry. Observation 2 suggests an answer: build a *toolkit* of privacy-preserving distributed computation techniques, that can be *assembled* to solve specific real-world problems. If such component assembly can be simplified to the point where it qualifies as development rather than research, practical use of privacy-preserving distributed data mining will become widely feasible.

This paper presents some early steps toward building such a toolkit. In Section 2 we describe several privacy-preserving computations. Section 3 shows several instances of how these can be used to solve privacy-preserving distributed data mining problems. We will also present promising future directions: tools that are needed, problems to be solved, and how to take this from research into development. First, though, we will discuss a formalism that enables us to capture and analyze what is meant by *privacy-preserving* distributed data mining.

1.1 Secure Multiparty Computation

The concept of *Secure Multiparty Computation* was introduced in [18]. The basic idea of Secure Multiparty Com-

putation is that a computation is secure if at the end of the computation, no party knows anything except its own input and the results. One way to view this is to imagine a trusted third party – everyone gives their input to the trusted party, who performs the computation and sends the results to the participants. Now imagine that we can achieve the same result without having a trusted party. Obviously, some communication between the parties is required for any *interesting* computation – how do we ensure that this communication doesn't disclose anything? The answer is to allow non-determinism in the exact values sent in the intermediate communication (e.g., encrypt with a randomly chosen key), and demonstrate that a party with just its own input and the result can generate a "predicted" intermediate computation that is as likely as the actual values. This has been shown possible[18; 9], however the general method given does not scale well to data mining sized problems.

A detailed discussion of Secure Multiparty Computation is given elsewhere in this issue[14], and we encourage readers who want a deep understanding of the following material to start with that article. We now give some examples of privacy preserving computations, show some of the subtleties involved in ensuring that such a computation is truly secure.

2. TECHNIQUES

We present here four efficient methods for privacy-preserving computations that can be used to support data mining. Not all are truly secure multiparty computations – in some, information other than the results is revealed – but all do have provable bounds on the information released. In addition, they are *efficient*: the communication and computation cost is not significantly increased through addition of the privacy preserving component.

This is by no means an exhaustive list of efficient secure multiparty computations. Some other examples can be found in [3; 6]. They are sufficient, however, to allow us to present several privacy-preserving solutions to data mining problems in Section 3.

2.1 Secure Sum

Secure sum is often given as a simple example of secure multiparty computation[16]. We include it here because of its applicability to data mining (see Sections 3.1 and 3.3), and because it demonstrates the difficulty and subtlety involved in making and proving a protocol secure.

Distributed data mining algorithms frequently calculate the sum of values from individual sites. Assuming three or more parties and no collusion, the following method securely computes such a sum.

Assume that the value $v = \sum_{l=1}^{s} v_l$ to be computed is known to lie in the range $[0..n]$.

One site is designated the *master* site, numbered 1. The remaining sites are numbered $2..s$. Site 1 generates a random number R, uniformly chosen from $[0..n]$. Site 1 adds this to its local value v_1, and sends the sum $R + v_1 \bmod n$ to site 2. Since the value R is chosen uniformly from $[1..n]$, the number $R + v_1 \bmod n$ is also distributed uniformly across this region, so site 2 learns nothing about the actual value of v_1.

For the remaining sites $l = 2..s-1$, the algorithm is as follows. Site l receives

$$V = R + \sum_{j=1}^{l-1} v_j \bmod n.$$

Since this value is uniformly distributed across $[1..n]$, i learns nothing. Site i then computes

$$R + \sum_{j=1}^{l} v_j \bmod n = (v_j + V) \bmod n$$

and passes it to site $l+1$.

Site s performs the above step, and sends the result to site 1. Site 1, knowing R, can subtract R to get the actual result. Note that site 1 can also determine $\sum_{l=2}^{s} v_l$ by subtracting v_1. This is possible from the global result *regardless of how it is computed*, so site 1 has not learned anything from the computation. Figure 1 depicts how this method operates.

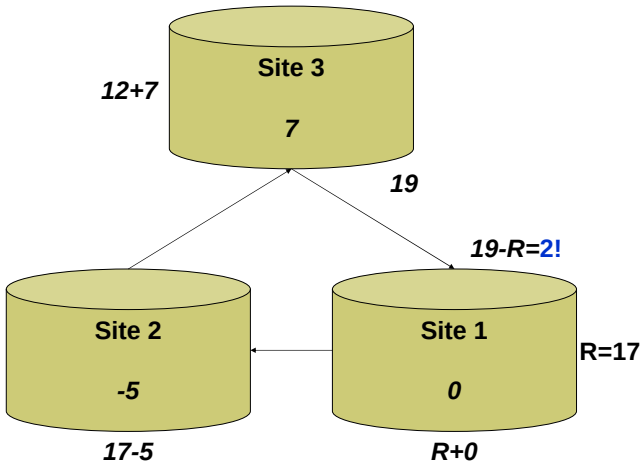

Figure 1: Secure computation of a sum.

This method faces an obvious problem if sites collude. Sites $l-1$ and $l+1$ can compare the values they send/receive to determine the exact value for v_l. The method can be extended to work for an honest majority. Each site divides v_l into shares. The sum for each share is computed individually. However, the path used is permuted for each share, such that no site has the same neighbor twice. To compute v_l, the neighbors of l from each iteration would have to collude. Varying the number of shares varies the number of dishonest (colluding) parties required to violate security.

2.2 Secure Set Union

Secure union methods are useful in data mining where each party needs to give rules, frequent itemsets, etc., without revealing the owner. The union of items can be evaluated using SMC methods if the domain of the items is small. Each party creates a binary vector where 1 in the i^{th} entry represents that the party has the i^{th} item. After this point, a simple circuit that *or's* the corresponding vectors can be built and it can be securely evaluated using general secure multi-party circuit evaluation protocols. However, in data mining the domain of the items is usually large. To overcome this problem a simple approach based on commutative encryption is used. An encryption algorithm is commutative if given encryption keys $K_1, \ldots, K_n \in K$, for any m in do-

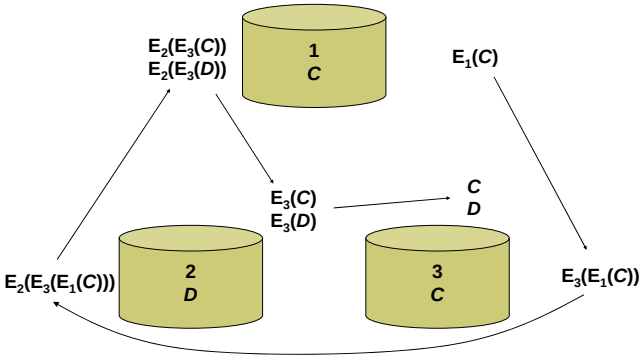

Figure 2: Determining the Union of a set of items.

main M, and for any permutation i, j, the following two equations hold:

$$E_{K_{i_1}}(\ldots E_{K_{i_n}}(M)\ldots) = E_{K_{j_1}}(\ldots E_{K_{j_n}}(M)\ldots) \quad (1)$$

$\forall M_1, M_2 \in M$ such that $M_1 \neq M_2$ and for given k, $\epsilon < \frac{1}{2^k}$

$$Pr(E_{K_{i_1}}(\ldots E_{K_{i_n}}(M_1)\ldots) = E_{K_{j_1}}(\ldots E_{K_{j_n}}(M_2)\ldots)) < \epsilon \quad (2)$$

With shared p the Pohlig-Hellman encryption scheme[15] satisfies the above equations, but any other commutative encryption scheme can be used.

The main idea is that each site encrypts its items. Each site then encrypts the items from other sites. Since equation 1 holds, duplicates in the original items will be duplicates in the encrypted items, and can be deleted. (Due to equation 2, only the duplicates will be deleted.) In addition, the decryption can occur in any order, so by permuting the encrypted items we prevent sites from tracking the source of an item. The algorithm for evaluating the union of the items is given in Algorithm 1, and an example is shown in Figure 2.

Clearly algorithm 1 finds the union without revealing which item belongs to which site. It is not, however, secure under the definitions of secure multi-party computation. It reveals the number of items that exist commonly in two sites, e.g. if k sites have an item in common, there will be an (encrypted) item duplicated k times. This does not reveal *which* items these are, but a truly secure computation (as good as each site giving its input to a "trusted party") could not reveal even this count. Allowing innocuous information leakage (the number of items that is owned by two sites) allows an algorithm that is sufficiently secure with much lower cost than a fully secure approach.

We can prove that other than the size of intersections and the final result, nothing is revealed. By assuming that the count of duplicated items is part of the final result, a Secure Multiparty Computation proof is possible.

2.3 Secure Size of Set Intersection

Consider several parties having their own sets of items from a common domain. The problem is to securely compute the cardinality/size of the intersection of these local sets.

Formally, given k parties $P_1 \ldots P_k$ having local sets $S_1 \ldots S_k$, we wish to securely compute $|S_1 \cap \cdots \cap S_k|$. We can do this is using a parametric commutative one way hash function. One way of getting such a hash function is to use commutative public key encryption, such as Pohlig Hellman, and

Algorithm 1 Finding secure union of items

Require: N is number of sites and Union_set = $\emptyset$ initially
{Encryption of all the rules by all sites}
for each site i **do**
  for each $X \in S_i$ **do**
    $M = newarray[N]$;
    $Xp = encrypt(X, e_i)$;
    $M[i] = 1$;
    Union_set $\bigcup$ (Xp,M);
  end for
end for{Site i encrypts its items and adds them to the global set. Each site then encrypts the items it has not encrypted before}

for each site i **do**
  for each tuple (r,M) $\in$ Union_set **do**
    if M[i] == 0 **then**
      rp=encrypt(r,e_i);
      M[i]=1;
      Mp= M ;
      Union_set=(Union_set-{(r,M)}) $\bigcup$ {(rp,Mp)};
    end if
  end for
end for

for (r,M) $\in$ Union_set and (rp,Mp) $\in$ Union_set **do** {check for duplicates}
  if r==rp **then**
    Union_set= Union_set-{(r,M)} {Eliminate duplicate items before decrypting};
  end if
end for

for each site i **do** {Each site decrypts every item to get the union of items}
  for all (r,M) $\in$ Union_set **do**
    rd = decrypt(r,d_i) ;
    Union_set=(Union_set-{(r,M)}) $\bigcup$ {(rd)};
  end for
  permute elements in the Union_set
end for
return Union_set

throw away the decryption keys. Commutative encryption has already been described in detail in Section 2.2.

All k parties locally generate their public key-pair (E_i, D_i) for a commutative encryption scheme. (They can throw away their decryption keys since these will never be used.) Each party encrypts its items with its key and passes it along to the other parties. On receiving a set of (encrypted) items, a party encrypts each item and permutes the order before sending it to the next party. This is repeated until every item has been encrypted by every party. Since encryption is commutative, the resulting values from two different sets will be equal if and only if the original values were the same (i.e., the item was present in both sets). Thus, we need only count the number of values that are present in *all* of the encrypted itemsets. This can be done by any party. None of the parties is able to know which of the items are present in the intersection set because of the encryption. The complete protocol is shown in algorithm 2.

Algorithm 2 Securely computing size of intersection set

Require: k sites
Require: each site has a local set S_i
Generate the commutative encryption key-pair (E_i, D_i)
{Throw away the decryption keys, since they will not be needed.}
$M = S_i$
for $k-1$ steps **do**
$M' = newarray[|M|]$
j=0;
for each $X \in M$ **do**
$M'[j++] = encrypt(X, E_i)$
end for
permute the array M' in some random order
send the array M' to site $i+1 \mod k$
receive array M from site $i-1 \mod k$
end for
$M' = newarray[|M|]$
j=0;
for each $X \in M$ **do**
$M'[j++] = encrypt(X, E_i)$
end for
permute the array M' in some random order
send M' to site $i \mod 2$ {This prevents a site from seeing it's own encrypted items}
sites 0 and 1 produce array I_0 and I_1 containing only (encrypted) items present in all arrays received.
site 1 sends I_1 to site 0
site 0 broadcasts the result $|I_0 \cup I_1|$

2.4 Scalar Product

Scalar product is a powerful component technique. Many data mining problems can essentially be reduced to computing the scalar product. One example of this, reducing association rule mining to scalar product computation, will be discussed in Section 3.2. The problem can be formally defined as follows: Assume 2 parties P_1 and P_2 each have a vector of cardinality n; i.e. P_1 has $\vec{X} = (x_1 \dots x_n)$ and P_2 has $\vec{Y} = (y_1 \dots y_n)$. The problem is to securely compute the scalar product of the two vectors, i.e., $\sum_{i=1}^{n} x_i * y_i$. Recently, there has been a lot of research into this problem, which has given rise to many different solutions with varying degrees of accuracy, communication cost and security[3; 11; 17]. Note that all of these techniques are limited to the 2-party version of the problem and cannot easily be extended to the general case. In [3] the problem is modeled as Secure Multiparty Computation and the present a solution using cryptographic techniques (oblivious transfer). This, however, is not very efficient. The key insight in [17] is to use linear combinations of random numbers to disguise vector elements and then do some computations to remove the effect of these randoms from the result. The solution is briefly explained in algorithm 3. Though this method does reveal more information than just the input and the result, it is efficient and suited for large data sizes, thus being useful for data mining.

Algorithm 3 Computing the scalar product

Require: N=2 is number of sites, site A and site B
Require: Each site has a vector of cardinality n. Thus,
Require: A has $\vec{X} = (x_1, \dots, x_n)$ and
Require: B has $\vec{Y} = (y_1, \dots, y_n)$
Both A and B together decide on a random $n \times n/2$ matrix C
for Site A **do**
A generates a random vector R of cardinality $n/2$ ($\vec{R} = R_1, \dots, R_{n/2}$)
A generates the $n \times 1$ addition matrix X' by multiplying C with R (i.e. $X' = C \times R$)
A generates $X'' = X + X'$
A sends X'' to B
end for{First message from A to B}
for Site B **do**
B generates the scalar product S' of X'' and Y (i.e. $S' = \sum_{i=1}^{n} x''_i * y_i$)
B also generates the $n \times 1$ matrix $Y' = C^T \times Y$
B sends both S' and Y' to A
end for{First message from B to A}
for Site A **do**
A generates the subtraction factor $S'' = \sum_{i=1}^{n} Y'_i * R_i$
A generates the required scalar product $S = S' - S''$
A reports the scalar product S to B
end for{Second message from A to B}

While the general m-party case of the scalar product is an easily stated problem, none of the solutions suggested so far can be easily extended to solve the general problem.

3. APPLICATIONS

We now demonstrate how the above protocols can be used to make several standard data mining algorithms into privacy-preserving distributed data mining algorithms.

3.1 Association rules in horizontally partitioned data

We address association rule mining as defined in [1]: Let $I = \{i_1, i_2, \dots, i_n\}$ be a set of items and DB be a set of transactions, where each transaction $T \in DB$ is an itemset such that $T \subseteq I$. Given an itemset $X \subseteq I$, a transaction T *contains* X if and only if $X \subseteq T$. An association rule is an implication of the form $X \Rightarrow Y$ where $X \subseteq I, Y \subseteq I$ and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ has *support* s in the transaction database DB if s% of the transactions in DB contain $X \cup Y$. The rule has *confidence* c if c% of the transactions in DB

that contain X also contains Y. An itemset X with k items is called a k-itemset. The problem of mining association rules is to find all rules whose support and confidence are higher than a specified minimum support and confidence.
In a horizontally partitioned database, the transactions are distributed among n sites. The global support count of an item set is the *sum* of all the local support counts. An itemset X is *globally supported* if the global support count of X is bigger than s% of the total transaction database size. The global confidence of a rule $X \Rightarrow Y$ can be given as $\{X \cup Y\}.sup/X.sup$. A k-itemset is called a globally large k-itemset if it is globally supported.
The aim of distributed association rule mining is to find all rules whose global support and global confidence are higher than the user specified minimum support and confidence.
The FDM algorithm [4] is a fast method for distributed mining of association rules. We summarize this below:

1. **Candidate Set Generation**: Intersect the globally large itemsets of size $k-1$ with locally large $k-1$ itemsets to get candidates. From these, use the classic apriori candidate generation algorithm to get the candidate k itemsets.

2. **Local Pruning**: For each X in the local candidate set, scan the local database to compute the local support of X. If X is locally large, it is included in the locally large itemset list.

3. **Itemset Exchange**: Broadcast locally large itemsets to all sites – the *union* of locally large itemsets, a superset of the possible global frequent itemsets. (It is clear that if X is supported globally, it will be supported at least at one site.) Each site computes (using apriori) the support of items in union of the locally large itemsets.

4. **Support Count Exchange**: Broadcast the computed supports. From these, each site computes globally large k-itemsets.

The above algorithm avoids disclosing individual transactions, but does expose significant information about the rules supported at each site. Our goal is to approximate the efficiency of the above algorithm, without requiring that any site disclose its locally large itemsets, support counts or transaction sizes.
Our algorithm basically modifies the above outlined method. In the **Itemset Exchange** step, the secure union algorithm of Section 2.2 will be used to get the secure union. After this step, the globally supported itemsets can be easily found by a secure sum. Since the goal is to determine if the support exceeds a threshold, rather than learn exact support, we alter the algorithm slightly. Instead of sending $R + \sum v_i$ to site 1, site k performs a secure comparison with site 1 to see if $R + \sum v_i \geq R$. If so, the support threshold is met. The confidence of large itemsets can also be found using this method. We would like to emphasize that if the goal is to have a totally secure method, the union step would have to be eliminated. However, using the secure union method gives higher efficiency with provably controlled disclosure of some minor information (i.e., the number of duplicate items and the candidate sets.) The validity of even this disclosed information can be reduced by noise addition. Basically, each site can add some fake large itemsets to its actual locally large itemsets. In the pruning phase, the fake items will be eliminated.
This gives a brief, oversimplified idea of how the method works. Full discussion can be found in [12].

3.2 Association rules in vertically partitioned data

Mining private association rules from vertically partitioned data, where the items are partitioned and each itemset is split between sites, can be done by extending the existing apriori algorithm. Most steps of the apriori algorithm can be done locally at each of the sites. The crucial step involves finding the support count of an itemset. If we can securely compute the support count of an itemset, we can check if the support is greater than threshold, and decide whether the itemset is frequent. Using this, we can easily mine association rules securely.
Consider the entire transaction database to be a boolean matrix where 1 represents the presence of that item (column) in that transaction (row), while 0 correspondingly represents an absence. The key insight is as follows: The support count of an itemset is *exactly* the scalar product of the vectors representing the sub-itemsets with both parties. Thus, if we can compute the scalar product securely, we can compute the support count. Full details are given in [17].
Another way of finding the support count is as follows: Let party i represent its sub-itemset as a set S_i which contains only those transactions which support the sub-itemset. Then the size of the intersection set of all these local sets $(|S|, S = \cap_{i=1}^{n} S_i)$, gives the support count of the itemset. This can be done using the protocol in Section 2.3.
These protocols assume a semi-honest model, where the parties involved will honestly follow the protocol but can later try to infer additional information from whatever data they receive through the protocol. One result of this is that parties are not allowed to give spurious input to the protocol. If a party is allowed to give spurious input, they can probe to determine the value of a specific item at other parties. For example, if a party gives the input $(0, \ldots, 0, 1, 0, \ldots, 0)$, the result of the scalar product (1 or 0) tells the malicious party if the other party the transaction corresponding to the 1. Attacks of this type can be termed probing attacks. All of the protocols currently suggested in the literature are susceptible to probing attacks. Better techniques which work even in the malicious model are needed to guard against this.

3.3 EM Clustering

We present a privacy preserving EM algorithm for secure clustering. Only the one dimensional case is shown; extension to multiple dimensions is straight forward. The convention for notations of the paper is given in Table 1 while i, j, l are the indexes for the mixture component, data points and distributed sites respectively. t denotes the iteration step.
From conventional EM mixture models for clustering, we assume that data y_j are partitioned across s sites $(1 \leq l \leq s)$. Each site has n_l data items, where summation over all the sites gives n. To obtain a global estimation for $\mu_i^{(t+1)}$, $\sigma_i^{2(t+1)}$, and $\pi_i^{(t+1)}$ (the E step) requires only the global

Table 1: Convention for symbols.

k	Total number of mixture components (clusters).
s	Total number of distributed sites.
n	Total number of data points.
n_l	Total number of data points for site l.
y_j	Observed data points for the one dimensional case.
μ_i	Mean for cluster i in the one-dimensional case.
σ_i^2	Variance for cluster i in the one dimensional case.
π_i	Estimate of proportion of items in cluster i.
z_{ij}	Cluster membership. If $y_j \in$ cluster i, $z_{ij} \approx 1$, else $z_{ij} \approx 0$.

values n and

$$\sum_{j=1}^{n} z_{ij}^{(t)} y_j = \sum_{l=1}^{s}\sum_{j=1}^{n_l} z_{ijl}^{(t)} y_j \quad (3)$$

$$\sum_{j=1}^{n} z_{ij}^{(t)} = \sum_{l=1}^{s}\sum_{j=1}^{n_l} z_{ijl}^{(t)} \quad (4)$$

$$\sum_{j=1}^{n} z_{ij}^{(t)} (y_j - \mu_i^{(t+1)})^2 = \sum_{l=1}^{s}\sum_{j=1}^{n_l} z_{ijl}^{(t)} (y_j - \mu_i^{(t+1)})^2 \quad (5)$$

Observe that the second summation in each of the above equations is local:

$$A_{il} = \sum_{j=1}^{n_l} z_{ijl}^{(t)} y_j \quad (6)$$

$$B_{il} = \sum_{j=1}^{n_l} z_{ijl}^{(t)} \quad (7)$$

$$C_{il} = \sum_{j=1}^{n_l} z_{ijl}^{(t)} (y_j - \mu_i^{(t+1)})^2 \quad (8)$$

It is easy to see that sharing these values across sites does not reveal y_j. Furthermore, it is not necessary to share n_l, A_{il}, B_{il}, and C_{il}, but only to compute the global sums. Section 2.1 shows how to compute these summations securely.

The estimation step giving $\mathbf{z}$ can be partitioned and computed locally given global μ_i, σ_i^2, and π_i:

$$z_{ijl}^{(t+1)} = \frac{\pi_i^{(t)} f_i(y_j; \mu_i^{(t)}, \sigma_i^{2(t)})}{\sum_i \pi_i^{(t)} f_i(y_j; \mu_i^{(t)}, \sigma_i^{2(t)})} \quad (9)$$

where y_j is a data point at site l. The E-step and M-step iterate until

$$|L^{(t+1)} - L^{(t)}| \leq \epsilon. \quad (10)$$

where

$$L^{(t)}(\theta^{(t)}, \mathbf{z}^{(t)}|y) = \sum_{j=1}^{n}\sum_{i=1}^{k} \{z_{ij}^{(t)} [\log \pi_i f_i(y_j^{(t)}|\theta^{(t)})]\}. \quad (11)$$

Algorithm 4 summarizes the method.

Algorithm 4 Secure EM Algorithm.

At each site l, $\forall_{i=1..n_l, j=1..k}$ randomly initialize z_{ijl} to 0 or 1.
Use secure sum of Section 2.1 to compute $n = \sum_{l=1}^{s} n_l$
$t = 0$
while Threshold criterion of log likelihood not met **do**
  for all $i = 1..k$ **do**
    At each site l, calculate $A_{il}^{(t+1)}$ and $B_{il}^{(t+1)}$ using equations (6) and (7).
    Use secure sum to calculate $A_i^{(t+1)}$ and $B_i^{(t+1)}$.
    Site 1 uses these to compute $\mu_i^{(t+1)}$ and broadcasts it to all sites.
    Each site l calculates $C_{il}^{(t+1)}$ using equation (8).
    Use secure sum to calculate $C_i^{(t+1)}$.
    Site 1 calculates $\sigma_i^{2(t+1)}$ and $\pi_i^{(t+1)}$ and broadcasts them to all sites.
    At each site l, $\forall_{j=1..n_l}$ update $z_{ijl}^{(t+1)}$ using equation (9).
  end for
  $t = t + 1$
  Calculate the log likelihood difference using equation (10) and (11).
end while

4. CONCLUDING REMARKS

We believe it is feasible to construct a toolkit of privacy preserving computations that can be used to build data mining techniques. As we have seen, there are still many subtleties involved – simply performing one secure computation, then using those results to perform another, reveals intermediate information that is not part of the final results. The resulting data mining technique no longer meets the definition of a secure multiparty computation – how can we state that it is privacy-preserving? One solution is to define the intermediate information as part of the "results", enabling a secure multiparty computation proof that nothing *else* is revealed. Then we can evaluate if the real results and the extra intermediate information violate privacy constraints – if not, the resulting technique is sufficient. This gives us the ability to guarantee *controlled disclosure*.

This becomes more difficult with iterative techniques. The intermediate results from several iterations may reveal a lot of information; showing that this does not violate privacy may be difficult or impossible. The general approach of [18] may provide a solution: split the intermediate results into randomly-determined "shares", combining the shares only at the end of the computation. Still open is to determine if data mining techniques can be partitioned so that independently computed shares of a result will converge to results that when combined give the desired global result.

We are also working on a framework generalized from [2], which uses distortion to protect data privacy and still estimate the original distribution. Assume we are calculating function f from site 1 containing data X and site 2 containing data Y. X and Y are contaminated by ϵ_1 and ϵ_2 respectively. It is now a classical optimization problem where we would like to find ϵ_1 and ϵ_2 such that:

$$E\|f(X + \epsilon_1, Y + \epsilon_2) - f(X, Y)\|$$

is minimized while $Var(\epsilon_1)$ and $Var(\epsilon_2)$ are maximized. Generalizing into this framework will allow us to better assess the level of privacy and security of a protocol and build analytical results for each f of interest.

We are continuing work in this area, both to develop new tools for building privacy preserving data mining techniques, and to demonstrate new applications for this technology. There are still many challenges in this area, such as defining

privacy constraints[5]. As an example of the potential difficulties, imagine a scenario where the data mining results violate privacy. Secure multiparty computation definitions do not solve the problem. We look forward to seeing expanded research in this field.

5. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, Santiago, Chile, Sept. 12-15 1994. VLDB.

[2] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD Conference on Management of Data*, pages 439–450, Dallas, TX, May 14-19 2000. ACM.

[3] M. J. Atallah and W. Du. Secure multi-party computational geometry. In *Seventh International Workshop on Algorithms and Data Structures (WADS 2001)*, Providence, Rhode Island, USA, Aug. 8-10 2001.

[4] D. W.-L. Cheung, J. Han, V. Ng, A. W.-C. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. In *Proceedings of the 1996 International Conference on Parallel and Distributed Information Systems (PDIS'96)*, Miami Beach, Florida, USA, Dec. 1996.

[5] C. Clifton, M. Kantarcioglu, and J. Vaidya. Defining privacy for data mining. In H. Kargupta, A. Joshi, and K. Sivakumar, editors, *National Science Foundation Workshop on Next Generation Data Mining*, pages 126–133, Baltimore, MD, Nov. 1-3 2002.

[6] W. Du and M. J. Atallah. Privacy-preserving cooperative scientific computations. In *14th IEEE Computer Security Foundations Workshop*, pages 273–282, Nova Scotia, Canada, June 11-13 2001.

[7] A. Eisenberg. With false numbers, data crunchers try to mine the truth. *New York Times*, July 18 2002.

[8] S. Evfimievski. Randomization techniques for privacy-preserving association rule mining. *SIGKDD Explorations*, 4(2), Dec. 2002.

[9] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game - a completeness theorem for protocols with honest majority. In *19th ACM Symposium on the Theory of Computing*, pages 218–229, 1987.

[10] M. Hamblen. Privacy algorithms: Technology-based protections could make personal data impersonal. *Computerworld*, Oct. 14 2002.

[11] I. Ioannidis, A. Grama, and M. Atallah. A secure protocol for computing dot-products in clustered and distributed environments. In *The 2002 International Conference on Parallel Processing*, Vancouver, British Columbia, Aug. 18-21 2002.

[12] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *The ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'02)*, pages 24–31, June 2 2002.

[13] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Advances in Cryptology – CRYPTO 2000*, pages 36–54. Springer-Verlag, Aug. 20-24 2000.

[14] B. Pinkas. Cryptographic techniques for privacy-preserving data mining. *SIGKDD Explorations*, 4(2), Dec. 2002.

[15] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, IT-24:106–110, 1978.

[16] B. Schneier. *Applied Cryptography*. John Wiley & Sons, 2nd edition, 1995.

[17] J. S. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 639–644, July 23-26 2002.

[18] A. C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167. IEEE, 1986.