Classification Spanning Private Databases

Ke Wang[+], Yabo Xu[+], Rong She[+], Philip S. Yu[*]

[+] School of Computing Science, Simon Fraser University
8888 University Drive, Burnaby, BC V5A 1S6, Canada
[+] wangk,yxu,rshe@cs.sfu.ca

[*] IBM T.J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532, USA
[*] psyu@us.ibm.com

Abstract

In this paper, we study the classification problem involving information spanning multiple private databases. The privacy challenges lie in the facts that data cannot be collected in one place and the classifier itself may disclose private information. We present a novel solution that builds the same decision tree classifier as if data are collected in a central place, but preserves the privacy of participating sites.

Introduction

As one of the most effective classifiers to date, decision trees are used in many applications such as spam filtering, fraud detection, medical diagnosis, text categorization, etc. Traditional decision trees require data in a single table and when information is scattered across multiple sources, a pre-processing step is needed to collect all relevant data into a single table. Today, this practice is challenged by the demand of analyzing information spanning multiple private databases. Let us consider an example.

Example 1. A Law Enforcement Agency (LEA) wants to detect insider trading. LEA owns historical data **L**(*SSN, Org, Class*), which records whether a person from certain organization was involved in insider trading. This data alone is not useful because a classifier based on individual's social security number and organization names tend to overfit the data. A better alternative is to examine whether the trading is in favor of the dealer by looking at other related data, say, the following tables:

Trading table: **S**(τ, *Dealer, Type, Stock, Company,Gain*)
Phone call table: **P**(τ, *Caller, Callee*)

where τ is the timestamp, "*Gain*" represents the "Up" and "Down" movement at the closing of the day. "*Type*" is either "Buy" or "Sell". Suppose LEA has access to these tables. The following join computes instances where a caller calls a dealer who subsequently trades:

SELECT *
FROM *P, S, L*
WHERE *L.SSN=P.Caller AND P.Callee=S.Dealer AND P.*τ*<S.*τ

The resulted joined table may introduce rules that involve attributes from multiple source tables, for example:

((*Gain*="Up" AND *Type*="Buy") OR (*Gain*="Down" AND *Type*="Sell")) AND (*Org*=Company) → *Class*=Yes.

That is, after getting a call (i.e., a hint), the trading on caller's company stock is favorable (i.e., a "Sell" transaction precedes a "Down" movement; or "Buy" precedes "Up" movement). Such patterns spanning multiple tables are more useful for future prediction because it refers to "behaviors", not "identities". ■

Driven by today's trends of end-to-end integration, business outsourcing, simultaneous competition and cooperation, privacy and security, there is an increasing demand for analyzing integrated information across private databases. As data cannot be collected into a central place, traditional methods have difficulties on classifier construction. This motivates our work in following aspects:

- **Privacy preserving collaboration**. While the training space often requires the join of data from multiple distributed sources, privacy concerns place a major restriction on sharing information across organizations. In the above example, it may be difficult for LEA to have absolutely free access to the entire trading and phone call databases owned by other organizations. The traditional "join-then-mine" approach violates the privacy constraint. The classifier itself also contains private information. How to benefit from the classifier without leaking private information from it has not been explored.
- **Blow-up of the training space**. The number of records in the joined table blows up due to many-to-many join relationships, and the traditional "join-then-mine" suffers from this blow-up. E.g., if Bob is called x times and trades y times, the join condition P.Callee=S.Dealer will generate x*y join records involving Bob. With such unbounded blow-ups, the scalability of an algorithm should be measured relative to the size of source tables, not the joined table. Note the techniques of reducing training space by feature selection or sampling are difficult to apply here due to privacy issues and distributed nature of the data.

In this paper, we propose *secure join classification* as a way to integrate private databases for classification. The training space is specified, but not computed, by a join over several private databases. We present a solution in the form of decision tree which addresses both privacy concerns and scalability issues. Our goal is to build the decision tree identical to the one built as if all private tables were joined first, with the constraint that no private information is revealed to other participating parties.

Related Work

Privacy preserving classification was proposed in (Agrawal and Srikant 2000). Their idea is to perturb individual data items while preserving data distribution at an aggregated level. In (Lindell and Pinkas 2000), they deal with *horizontally* partitioned data where the sites do not

necessarily need to exchange data records and propose a solution using the oblivious transfer protocol. A decision tree algorithm for *vertically* partitioned data was presented in (Du and Zhan 2002), where they use a scalar product protocol to compute information gain at a decision tree node. This technique requires a common key among all tables. We do not assume such a common key.

Building decision trees on the join of multiple tables has been studied in (Wang et. al 2005). However, the focus was on scalability and no privacy issues were involved. On the other hand, while (She et. al 2005) discussed classification on the join of private tables, it addresses the problem by removing irrelevant features and does not guarantee strict privacy.

To the best of our knowledge, privacy breaches through the construction and use of decision tree have not been considered before. All previous works assume the decision tree can be freely accessed by all participating sites. In practice, this may not be reasonable because private information can be contained in the decision tree through split criteria along a root-to-leaf path. In fact, substantially more information can be disclosed through a larger intermediate tree before it is pruned. Even with pre-pruning technique (Rastogi and Shim 1998), a larger intermediate tree will be created. Thus the site keeping the decision tree learns more than the final decision tree. We will address this loophole.

Problem Statement

We consider k tables $T_1,...,T_k$, distributed among k parties so that each T_i is a private table at a different site. Without loss of generality, we assume T_1 is the *target table* containing the class column, with the domain of $C_1,...,C_u$. The set of *training instances* for classification is specified by a join query over $T_1,...,T_k$, which is a conjunction of predicates $T_i.A=T_j.B$ ($i\neq j$), where $T_i.A$ and $T_j.B$, called *join attributes,* represent one or more attributes from T_i and T_j. There is at most one such predicate between each pair of T_i and T_j. We define *join graph* such that there is an edge between T_i and T_j if there is a predicate $T_i.A=T_j.B$. We consider join queries for which the join graph is acyclic and connected, i.e., a tree. In practice, many database queries are indeed acyclic, e.g., chain joins and star joins over the star/snowflake schemas (Levene and Loizou 2003).

Definition 1. Given private tables $T_1,...,T_k$, *secure join classification* builds the classifier such that: (1) *training instances* are specified by a join query over all tables; (2) no site learns private information about other sites. ■

Note although the training space is defined by the join, the input is given as k source tables, not one joined table. Also, *dangling records* that do not contribute in the join are not included in the training space. Note we do not assume dangling records are removed beforehand; in fact, the removal of dangling records is not straightforward due to privacy issues and details will be discussed later.

We assume all sites to be honest, curious, but not malicious, as in (Agrawal, Evfimievski and Srikant 2003) for secure join and intersection. Our privacy model can be described by three types of attributes in each table:

- *Non-private* class column: the class column can be revealed to all sites, as in (Du and Zhan 2002). When all sites collaborate to build a classifier, it is reasonable to assume they are willing to share class information.
- *Semi-private* join attributes: with a predicate $T_i.A=T_j.B$, the join attributes $T_i.A$ and $T_j.B$, are semi-private in that the *shared* join values, i.e., $T_i.A\cap T_j.B$, can be revealed to each other, but the *non-shared* values, i.e., $(T_i.A\cup T_j.B)$-$(T_i.A\cap T_j.B)$, cannot. This was also adopted in (Agrawal, Evfimievski and Srikant 2003).
- *Private* non-join attributes: values of all non-join attributes cannot be revealed to any other sites.

Essentially, any values known to both joining sites are not private, but everything else is. Such privacy requirements must be enforced at all times, during both the construction and the use of the classifier. Nevertheless, the classifier should have exactly the same performance as constructed from the joined table.

Secure Decision Tree

We propose *secure decision tree (SDT)* to address privacy issues. A privacy-preserving decision tree must protect privacy throughout its representation, construction and usage. In this section, we examine the SDT representation and usage. We will present details of SDT construction in the next section. In the rest of this paper, where there is no confusion, T_i may also refer to the owning site of table T_i.

Representation of SDT

Traditionally, at each internal node of a decision tree, records are split to child nodes by a *split criterion* in the form of "$A\ \theta\ v$", where A is an attribute, θ is an arithmetic operator, v is a value of attribute A, e.g., "age>18". Split criteria along a root-to-leaf path may contain private attribute values. With a larger intermediate tree, more private information may be leaked during tree construction.

In a SDT, the split criterion at each node is kept only at its *split site* (the site that owns the splitting attribute) and hidden from all other sites. An internal node contains only the *identifier* of the split site, not the split criterion itself. Each site keeps a *split list* in the form of *(node, split criterion)* to track nodes that it splits. Such SDT contains only identities of split sites without any private attributes or values, thus can be kept at any participating site or an honest-but-curious third-party site. This is different from requiring a trusted party to hold the standard decision tree that contains attribute values. We will call the site that holds the SDT "*the coordinator*".

Using SDT

The SDT can be used to classify a new instance t in the form of $<t_1,\ldots,t_k>$, where t_i is the sub-record in the domain of T_i. This instance is known to all sites as $<Id_1,\ldots,Id_k>$, where Id_i is the record identifier of sub-record t_i. Only the

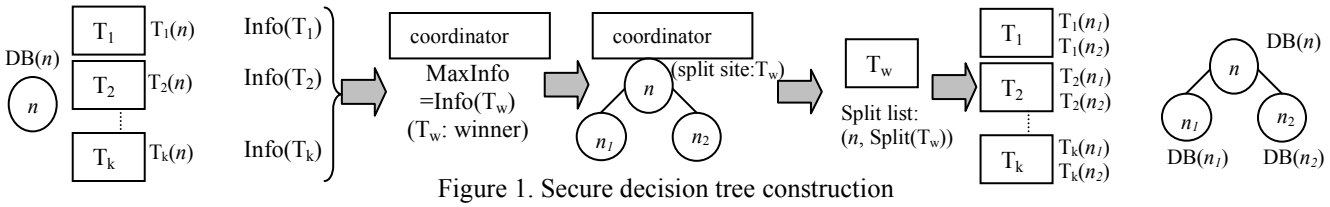

Figure 1. Secure decision tree construction

owner site T_i knows the private data in sub-record t_i. All sites first obtain a copy of SDT from the coordinator. Starting from the root node, each site checks if it is the split site at the current node. One and only one site will succeed. The succeeding site descends one branch based on the sub-record t_i. It then informs other sites to descend the same branch by specifying the branch number. This process is then repeated until a leaf node is reached and the class label recorded at the leaf node is announced. In the entire process, the only information exchanged among sites is the root-to-leaf path and the final class label. No private or semi-private data is ever leaked during the use of SDT.

Construction of Secure Decision Tree

Ignoring the privacy aspect, our approach builds *the exactly same* decision tree as "join-then-mine" does. In the "join-then-mine" approach, the joined data at each tree node n, denoted *Join(n)*, is recursively partitioned into *Join(n_1)* and *Join(n_2)* for child nodes n_1 and n_2,[1] until the data partition at a node consists entirely or dominantly of instances from one class. The split criterion is chosen to maximize information gain (Quinlan 1993) or gini index (Breiman et al. 1984), and the information needed for the calculation at node n is the class count for each attribute value in *Join(n)*, or the *AVC set* (*Attribute-Value-Class*) as called in (Gehrke, Ramakrishnan and Ganti 1998).

In a SDT, *Join(n)* cannot be used since data cannot be joined due to privacy concerns. Instead, the data partition at a node n is represented by a set of partitions on all tables, denoted as *DB(n)*. In particular, *DB(n)*={$T_1(n)$,..., $T_k(n)$}, where $T_i(n)$ is the partition on table T_i at node n and is *stored at its owner site*. Instead of splitting *Join(n)*, we split each $T_i(n)$ *at its owner site*. To ensure the same splits, *DB(n)* must provide the same AVC set as *Join(n)* does.

Definition 2. Let *Att*(T_i) denote the set of attributes in table T_i. For each class label C_j, we add a new column at each table to store the class count. (The new columns <C_1,...,C_u> are referred to as *class count matrix* or *Cls* in short.) We say *DB(n) models Join(n)* if every $T_i(n)$ has the property such that it is equal to the result of query:

SELECT *Att*(T_i), *SUM*(C_1),..., *SUM*(C_u)
FROM *Join(n)*
GROUP BY *Att*(T_i) ■

Essentially, each record in $T_i(n)$ represents all its occurrences in *Join(n)* by the aggregated *Cls*; therefore, the AVC set computed using $T_i(n)$'s will be the same as computed using *Join(n)*. Although such definition may be seen as an instance of a "SIP" in (John and Lent 1997), we only use it to define *the property of* $T_i(n)$, not to compute $T_i(n)$. Also, our focus is on privacy protection which is quite different from the focus of the SIP paper where no privacy issues were involved.

Figure 1 gives an overview on SDT construction. Suppose at a node n (initially the root), we have *DB(n)* that models *Join(n)*. Note *DB(n)* is a conceptual notation that consists of $T_i(n)$'s stored at each site. First, each site T_i computes information gain *Info*(T_i) and split criterion *Split*(T_i) based on its local $T_i(n)$. Next, by communicating with all sites, the coordinator identifies the winning site, i.e., the site with maximum information gain, labels n with the winner ID, creates child nodes n_1 and n_2, and informs the winner. Then, the winner updates its split list by inserting an entry for node n. Finally, each site splits its $T_i(n)$ between n_1 and n_2 so that *DB*(n_i) models *Join*(n_i).

There remains two key challenges: (1) initially, how to obtain *DB(root)* that models *Join(root)*; (2) suppose *DB(n)* models *Join(n)* at a node n, how to split each $T_i(n)$ to obtain *DB*(n_1) and *DB*(n_2) such that *DB*(n_i) models *Join*(n_i). Computing $T_i(n)$ is *not* straightforward since it must reflect the join of all tables while preserving privacy of each table.

Class Propagation

To obtain *DB(root)*, we "propagate" *Cls* from the target table to every other table. Initially, for the target table, C_j=1 if a record has the class label C_j, otherwise, C_j=0. The propagation proceeds in two phases. In Phase 1, *Cls* is propagated in a *depth-first traversal (DFT)* of the join graph, starting from the target table. The *Cls* in the last table visited reflects the join of all tables. In Phase 2, the *Cls* in the last table is propagated to every other table so that the *Cls* in every table reflects all joins. In the entire propagation process, the first time visit of a table is a *forward propagation*, and backtracking to a previously visited table is a *backward propagation*.

We define arithmetic operators on *Cls*'s: given an operator ø and two *Cls*'s: M_1=<C_1,...C_u> and M_2=<C_1',...,C_u'>, M_1øM_2=<C_1øC_1',...,C_uøC_u'>. Specially, 0/0 is 0. E.g., <2,4>+<1,2>=<3,6>; <2,0>/<1,0>=<2,0>.

Phase 1. Given a table T_i whose *Cls* is computed, consider the next table T_j in *DFT* with join predicate $T_i.A=T_j.B$.

Forward propagation $T_i \rightarrow T_j$: Let $T[x]$ denotes the set of records in table T whose join value is x. A record t in T_j with join value v will join with *all* records in $T_i[v]$, thus t should get the aggregated *Cls* over all $T_i[v]$ records.

Definition 3. The *class summary* from T_i to T_j, denoted $CS(T_i \rightarrow T_j)$, consists of entries (v, *ClsSum*) for each distinct

[1] For simplicity, we consider only binary splits, the generalization of k children is straightforward.

value v in $T_i.A$, where *ClsSum* is the summation of *Cls*'s over all records in $T_i[v]$. ■

The site T_i computes $CS(T_i \rightarrow T_j)$ and passes it to T_j. For each record t in T_j, $CS(T_i \rightarrow T_j)$ is looked up. If an entry ($t.B$, *ClsSum*) is found, the *Cls* of t is updated:

$t.Cls \leftarrow ClsSum$; (1)

otherwise, t is removed since it is a dangling record. In the same scan, T_j also computes $CS(T_j \rightarrow T_v)$, where T_v is the next table to be visited (if any).

Example 2. Figure 2 shows an example of forward propagation. The join relationships are indicated by arrows connecting the join attributes. Initially, the target table T_1 computes $CS(T_1 \rightarrow T_2)$={(a,<1,0>),(b,<1,2>)}, which is used by T_2 to update its *Cls*'s. E.g., $R_{21}.Cls$=<1,0> given by (a,<1,0>). R_{22} is removed as there is no entry for c. T_2 also computes $CS(T_2 \rightarrow T_1)$={(a,<2,0>),(b,<1,2>)}. ■

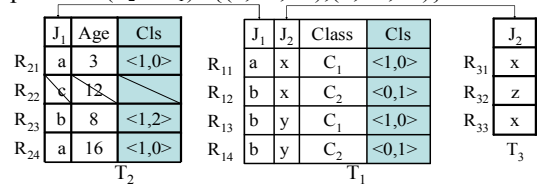

Figure 2. Forward propagation $T_1 \rightarrow T_2$

Backward propagation $T_i \rightarrow T_j$: For any join value v, *all* records in $T_j[v]$ will join with *all* records in $T_i[v]$. Note *Cls*'s of $T_j[v]$ records have been previously assigned in forward propagation, but they are outdated since they have not reflected the join performed after T_j was last visited. Thus given an entry (v, *ClsSum*) in $CS(T_i \rightarrow T_j)$, the *Cls* of $T_j[v]$ records should be adjusted such that: (1) for any $T_j[v]$ record, its share in $T_j[v]$ remains constant; (2) the aggregated *Cls* of $T_j[v]$ records must be equal to *ClsSum* that comes from $T_i[v]$.

Thus on receiving $CS(T_i \rightarrow T_j)$ from T_i, for each record t in T_j, if ($t.B$, *ClsSum*) is in $CS(T_i \rightarrow T_j)$, $t.Cls$ is rescaled:

$t.Cls \leftarrow t.Cls * ClsSum / ClsSum'$, (2)

where *ClsSum'* is the summation of *Cls*'s over all $T_j[t.B]$ records. Note ($t.B$, *ClsSum'*) is in the $CS(T_j \rightarrow T_i)$ computed previously in forward propagation. Otherwise t is dangling and is removed. T_j also computes $CS(T_j \rightarrow T_v)$ for the next table T_v (if any).

Example 3. Figure 3 shows backward propagation following Example 2. T_2 passes $CS(T_2 \rightarrow T_1)$ to T_1, where $R_{11}.Cls$ is rescaled to <1,0>*<2,0>/<1,0>=<2,0>, given (a,<2,0>) from $CS(T_2 \rightarrow T_1)$ and (a,<1,0>) from $CS(T_1 \rightarrow T_2)$. T_1 also computes $CS(T_1 \rightarrow T_3)$={(x,<2,1>),(y,<1,1>)} for forward propagation to T_3, who updates its *Cls* as in (1). ■

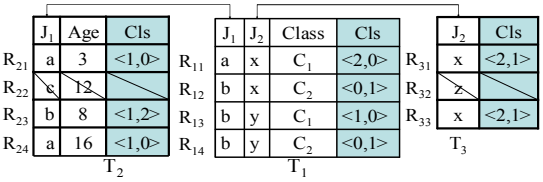

Figure 3. Backward propagation $T_2 \rightarrow T_1$

Phase 2. At the end of Phase 1, *Cls* in the last visited table has reflected all joins. Phase 2 adjusts *Cls* in other tables by backward propagation from that last table.

In the example above, at the end of phase 1, *Cls* in T_3 has reflected all joins, but not T_1 and T_2. Phase 2 includes backward propagations $T_3 \rightarrow T_1$ and $T_1 \rightarrow T_2$.

From the above discussion, we have:

Lemma 1. Let *DB(root)* denote the set of tables T_i at the end of Phase 2, then *DB(root)* models *Join(root)*. ■

Split Propagation

Suppose *DB(n)* models *Join(n)* at a node n, whose child nodes are n_1 and n_2. Each $T_i(n)$ needs to be split to create $DB(n_1)$ and $DB(n_2)$ that model $Join(n_1)$ and $Join(n_2)$. If T_i contains the splitting attribute, the split is determined by the split criterion stored in T_i. The split of any other table is done by propagation as follows.

Given a T_i that has been split into T_{i1} and T_{i2} for n_1 and n_2, consider a connected table T_j with join predicate $T_i.A=T_j.B$. For each record t in T_j, where $t.B=v$, let $ClsSum_1$ and $ClsSum_2$ denote the aggregated *Cls* over the records in $T_{i1}[v]$ and $T_{i2}[v]$, respectively. Since t joins with *every* record in both $T_{i1}[v]$ and $T_{i2}[v]$, t should appear in both $DB(n_1)$ and $DB(n_2)$ with $t.Cls$ being split into $t.Cls_1$ and $t.Cls_2$, following the split of $ClsSum_1$ and $ClsSum_2$:

$t.Cls_1 \leftarrow t.Cls*ClsSum_1/(ClsSum_1+ClsSum_2)$; (3)

$t.Cls_2 \leftarrow t.Cls*ClsSum_2/(ClsSum_1+ClsSum_2)$. (4)

Definition 4. The *split summary* from T_i to T_j, denoted $SS(T_i \rightarrow T_j)$, consists of (v,$ClsSum_1$,$ClsSum_2$) for each distinct value v in $T_i.A$, where $ClsSum_1$ and $ClsSum_2$ are summations of *Cls*'s over the records in $T_{i1}[v]$ and $T_{i2}[v]$. ■

Split propagation $T_i \rightarrow T_j$: T_i passes $SS(T_i \rightarrow T_j)$ to T_j. For each record t in T_j, if an entry ($t.B$,$ClsSum_1$,$ClsSum_2$) is in $SS(T_i \rightarrow T_j)$, $t.Cls_1$ and $t.Cls_2$ are computed as in (3) and (4). If $t.Cls_1$ is not "all-zero", add t with $t.Cls_1$ to T_{j1}. The same is done for $t.Cls_2$. T_j also computes AVC sets for n_1 and n_2, as well as $SS(T_j \rightarrow T_v)$ for next table T_v (if any).

Lemma 2. If *DB(n)* models *Join(n)*, then $DB(n_i)$ models $Join(n_i)$, where i=1,2. ■

It follows from Lemma 1 and Lemma 2 that:

Theorem 1. For every node n, *DB(n)* models *Join(n)*. ■

Algorithm Analysis

Privacy. As SDT contains no attribute values, the coordinator does not know attribute values owned by other sites. For non-join attributes, no site transmits their values *in any form* to other sites. For join attributes, some values are transmitted between joining sites through class/split summaries. Consider two sites T_i and T_j with join predicate $T_i.A=T_j.B$. $CS(T_i \rightarrow T_j)$ contains entries of the form (v, *ClsSum*), where v is a join value in $T_i.A \cup T_j.B$ and *ClsSum* contains the class information. The class column is non-private, so *ClsSum* itself does not pose a problem. $T_i.A$ and $T_j.B$ are semi-private, thus v can be exchanged between T_i and T_j only if $v \in T_i.A \cap T_j.B$. This can be done by first performing *secure intersection* (Agrawal, Evfimievski and Srikant 2003) between T_i and T_j to get $T_i.A \cap T_j.B$. Then

$CS(T_i \rightarrow T_j)$ can be modified to contain only entries for the join values in the intersection. As for $SS(T_i \rightarrow T_j)$, no modification is needed because all dangling records are removed at the end of class propagation. Additionally, when the coordinator needs to find the maximum $Info(T_i)$ among all sites, a secure protocol (Yao 1986) may be used to hide the actual $Info(T_i)$ values to further enforce privacy.

Privacy Claim. (1) No private attribute values are transmitted out of its owner site. (2) Semi-private attribute values are transmitted only between two joining sites and only if it is shared by both sites. ■

Scalability. Our algorithm is scalable in the following aspects. First, the *Cls* in source tables T_i allows us to have the "after-join" class information but deal with the "before-join" data size (i.e, Theorem 1). Second, the class/split summaries further aggregate *Cls*'s for all records having the same join value. These summaries have a size proportional to the number of distinct join values, not the number of records. Each probe into a summary takes constant time in either a hash or array implementation. Each edge in the join graph is traversed at most three times in class propagation and once in split propagation. Each traversal involves one scan of the destination table. Thus, the cost at a decision tree node *n* is proportional to the size of *DB(n)*, not the size of *Join(n)*. An additional cost is the secure intersection. However, it is performed only once for each pair of joining tables at the root node of SDT in class propagation and has no major impact.

Scalability Claim. At each decision tree node *n*, the cost is proportional to the size of *DB(n)*, not the size of *Join(n)*. ■

Empirical Evaluations

We need to verify following properties: (1) whether the formulation of join classification defines a better training space than the target table alone; (2) whether our algorithm scales up for data with large blow-up; (3) whether SDT is efficient in the distributed environment. We implemented two versions of our algorithms: *MultiTbl* for the centralized environment where all tables are kept at one site (without privacy issue); *SecureMultiTbl* for the distributed case where private tables are kept at different sites.

MultiTbl

To study scalability of our method in the centralized case, we compare *MultiTbl* with a scalable decision tree method *RainForest* (Gehrke, Ramakrishnan and Ganti 1998). *RainForest* operates on the joined table and *MultiTbl* does not require join. Both do not have privacy issues and run on a PC with 2GHz CPU/512MB memory/Win2000 Pro. We compared *MultiTbl* and *RainForest* on both real-life datasets and synthetic datasets. Two real-life datasets were used and only one is shown here due to space limit.

Mondial Dataset. This dataset is from the CIA World Factbook, the International Atlas and the TERRA database, used previously in (May 1999). We formulated the classification problem based on following 12 tables (showing only join attributes and class, *CID* is country ID):

Religion(CID,religion), Country(CID), City(CID), Politics(CID), EthnicGroup(CID), Language(CID), Population(CID), Economy(CID), Encompasses(CID, continent), IsMember(CID, org), Continent (continent), Organization(org).

The task is to predict a country's religion (*religion* is the class attribute). For simplicity, all religions with Christianity origin are grouped into class 1; all others are of class 2. By using the target table *Religion* alone, the classifier has only 60% accuracy. When all 12 tables are used (by equality join on common attributes), the accuracy is boosted to 97.7%, demonstrating the superiority of join classification. Note both *RainForest* and *MultiTbl* build the same decision tree, only *RainForest* requires join prior to tree construction. The join dramatically blows up the data size from 38KB to 54MB, i.e., over 1000 times.

Figure 4 compares *RainForest* and *MultiTbl*. The time for *RainForest* does not include the extra 103 seconds for joining all tables (using SQL Server 8.0). A data partition was cached if there is free memory or otherwise stored on disk. *MultiTbl* took less than 4 seconds when all partitions were on disk. Once the memory was increased to 10MB, it could keep all partitions in memory and took only 1 second. *RainForest* took 60 seconds when all partitions were on disk. Even when the memory size is 200MB, it still took more than 20 seconds. *MultiTbl* is a order of magnitude faster since it handles a much smaller dataset.

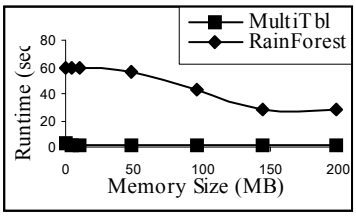

Figure 4. *MultiTbl* on Mondial dataset

Synthetic Datasets. As we are not aware of any existing synthetic datasets for our join classification problem, we designed our own data generator. We consider the chain join of *k* tables $T_1,\ldots,T_k$, where T_1 is the target table and each adjacent pair (T_i,T_{i+1}) have a join predicate. All join attributes have the same domain of size *V*. All tables contain *N* number of records. All tables have *X* numeric and *X* categorical attributes (except join attributes and class column). All numeric attributes have the ranked domain $\{1,2,\ldots,m\}$, where $m=\min\{N,10000\}$. Categorical values are drawn randomly from a domain of size 20. The class label in the joined table indicates whether at least half of numeric attributes have values in the top half of its domain.

Join values. Each table consists of *V* groups and the *jth group* uses the *j*th join value. Thus all records in the *j*th group of T_i, denoted $T_i\{j\}$, will join with exactly $T_{i+1}\{j\}$. The *jth join group* refers to the set of join records produced by the *j*th groups. The size S_j of the *j*th group is the same for all tables and determined by Poisson distribution with mean $\lambda=N/V$. The *j*th join group has the size S_j^k, with the

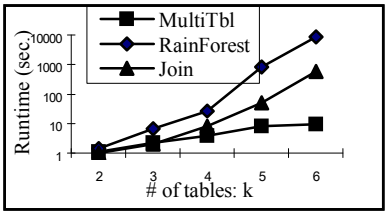

Figure 5. *MultiTbl* on synthetic datasets

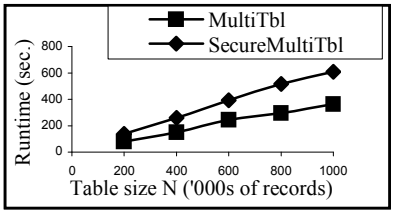

Figure 6. *SecureMultiTbl* overhead vs. table size

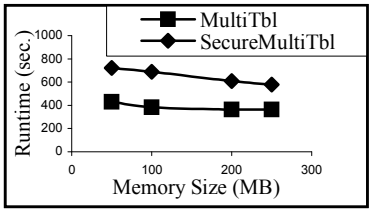

Figure 7. *SecureMultiTbl* overhead vs. memory size

mean λ^k. We control the blow-up of join through λ and k.

Numeric values. We generate numeric attributes such that all records in the jth join group have the same class, by having top half values in hj number of numerical attributes. To ensure this, we distribute hj among $T_1,\ldots,T_k$ randomly such that $hj=hj_1+\ldots+hj_k$ and all records in $T_i\{j\}$ have top half values in hj_i numerical attributes. hj follows uniform distribution in the range $[0,k*X]$, where $k*X$ is the total number of numerical attributes in a join record.

Class labels. If $hj \geq 0.5*k*X$, the class label is "Yes" for every record in $T_1\{j\}$; otherwise it's "No". On average, each group has 50% chance of having the "Yes" label.

The training and testing sets are independently generated. In all experiments, our accuracy ranged from 71% to 99%, much higher than the 50% guess based on class distribution. Figure 5 shows the time in log scale when the memory size is 200MB. N=1000, X=5, V=250, λ=4. "*Join*" is the additional time to join all tables as required by *RainForest*. The joined table size blows up with the number of tables. In the case of 6 tables, while *RainForest* suffers from the exponential blow-up, *MultiTbl* used only 6MB memory and is almost 1000 times faster.

SecureMultiTbl

We compared *SecureMultiTbl* with *MultiTbl* on synthetic datasets. *SecureMultiTbl* runs on 2 LAN-connected PCs: the first with 2GHz/512MB/Win2000 Pro; the second with 2.6GHz/1GB/WinXP. Each PC has 1 table with N varying from 200,000 to 1,000,000. X=5, λ=4, $V=N/\lambda$. *MultiTbl* is run on the first PC where both tables were stored. The comparison reveals the overhead for privacy preservation.

Figure 6 plots runtime vs. the source table size. The memory size is 200MB on each PC. *SecureMultiTbl* spent about an extra 60% of time on communications between PCs. The secure intersection took less than 5 seconds in all cases. Figure 7 plots runtime vs. memory size for the case of N=1,000,000. It shows that the additional overhead required by *SecureMultiTbl* is linear to the size of source tables, not the size of the joined table. Thus it is scalable in the distributed environment.

Conclusion

In today's globally distributed but connected world, classification often involves information from several private databases. The standard practice of joining all data into a single table does not address privacy concerns. Even the information disclosed through the classifier itself could damage privacy. We proposed "secure join classification" to address these issues and presented a concrete solution to build a secure decision tree. No private information is disclosed by either the tree construction process or the classifier itself. Our method runs in a time linear to the size of source tables and produces exactly the same decision tree as the unsecured "join-then-mine" approach.

Acknowledgments

This work is partially supported by a grant from NSERC.

References

R. Agrawal, A. Evfimievski and R. Srikant 2003. Information sharing across private databases. SIGMOD.

R. Agrawal and R. Srikant 2000. Privacy-preserving data mining. SIGMOD, 439–450.

L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone 1984. *Classification and Regression Trees*. Wadsworth: Belmont.

W. Du and Z. Zhan 2002. Building decision tree classifier on private data. ICDM Workshop on Privacy, Security and Data Mining.

J. Gehrke, R. Ramakrishnan and V. Ganti 1998. RainForest - A framework for fast decision tree construction of large datasets. VLDB.

G. H. John, B. Lent 1997. SIPping from the data firehose. KDD.

M. Levene and G. Loizou 2003. Why is the snowflake schema a good data warehouse design? *Information Systems* 28(3):225-240.

Y. Lindell and B. Pinkas 2000. Privacy preserving data mining. Advances in Cryptology-Crypto2000, Lecture Notes in Computer Science, Vol. 1880.

W. May 1999. Information extraction and integration with Florid: the Mondial case study. Technical Report, No. 131, Institut for Informatik, Universit at Freiburg. Data available at http://www.dbis.informatik.uni-goettingen.de/Mondial/.

J.R. Quinlan 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

R. Rastogi and K. Shim 1998. PUBLIC: A decision tree classifier that integrates building and pruning. VLDB.

R. She, K. Wang, Y. Xu and P. S. Yu 2005. Pushing feature selection ahead of join. SDM.

K. Wang, Y. Xu, P. S. Yu and R. She 2005. Building decision trees on records linked through key references. SDM.

A.C. Yao 1986. How to generate and exchange secrets. Annual Symposium on Foundations of Computer Sciences.