

Privacy preserving decision tree learning over multiple parties

F. Emekci *, O.D. Sahin, D. Agrawal, A. El Abbadi

Department of Computer Science, University of California at Santa Barbara, Santa Barbara, CA 93106, USA

Received 18 January 2006; received in revised form 14 April 2006; accepted 22 February 2007

Available online 28 March 2007

Abstract

Data mining over multiple data sources has emerged as an important practical problem with applications in different areas such as data streams, data-warehouses, and bioinformatics. Although the data sources are willing to run data mining algorithms in these cases, they do not want to reveal any extra information about their data to other sources due to legal or competition concerns. One possible solution to this problem is to use cryptographic methods. However, the computation and communication complexity of such solutions render them impractical when a large number of data sources are involved. In this paper, we consider a scenario where multiple data sources are willing to run data mining algorithms over the union of their data as long as each data source is guaranteed that its information that does not pertain to another data source will not be revealed. We focus on the classification problem in particular and present an efficient algorithm for building a decision tree over an arbitrary number of distributed sources in a privacy preserving manner using the ID3 algorithm.

Published by Elsevier B.V.

Keywords: Data mining; ID3; Data privacy and security

1. Introduction

In a changing world, data management does not only mean to store and retrieve data efficiently, but also derive meaningful information out of it. Recent advances in networking technologies enabled the collection and sharing of large amounts of data, which rendered distributed data mining an essential part of the data management. For example, enterprises have been using data mining algorithms such as association rule mining and decision tree learning for business logic. Most of the time, they make decisions based on the results of data mining algorithms over their data. Recent trends in global economy require enterprises to collaborate with each other to mine the union of their data to get more accurate and representative results for the market to be used in their business decisions. However, due to legal constraints or competition among enterprises, they do not want to reveal their data to other enterprises during the data mining process. The task of running data mining algorithms over multiple data sources without revealing any information other than the output of the algorithm to other sources is often referred to as *privacy preserving data mining*.

* Corresponding author.

E-mail addresses: fatih@cs.ucsb.edu (F. Emekci), odsahin@cs.ucsb.edu (O.D. Sahin), agrawal@cs.ucsb.edu (D. Agrawal), amr@cs.ucsb.edu (A. El Abbadi).

A straightforward solution for privacy preserving data mining is to use a trusted third party to gather data from all data sources and then send back results after running the desired data mining process. However, the level of trust is not acceptable in this scheme since the privacy of the data sources cannot be protected from the third party. There have been several approaches to support privacy preserving data mining over multiple private data sources [23,16,15,33,22] without using third parties. These solutions are usually based on cryptographic techniques such as commutative encryption and secure multi-party computations. Although these techniques are very secure, the excessive computation and communication cost associated render them impractical for scenarios involving a large number of parties. However, increasing the number of data sources usually results in more accurate and representative results since the training set is larger. For example, a decision tree built over data collected from a large number of enterprises will return more accurate predictions about the market.

In this paper, we propose a novel privacy preserving distributed decision tree learning algorithm that is based on the ID3 algorithm [27] and Shamir's secret sharing [30]. The proposed algorithm is scalable in terms of computation and communication cost, and therefore it can be run even when there is a large number of parties involved. Our goal, in particular, is to implement the ID3 algorithm for constructing decision trees over an arbitrary number of distributed data sources in a privacy preserving manner.

The rest of the paper is organized as follows. Related work is surveyed in Section 2. In Section 3, we provide some background information. Section 4 introduces a technique based on Shamir's secret sharing to calculate the summation of multiple secret values without revealing any other information. Section 5 builds on this technique to implement the ID3 algorithm over multiple data sources in a privacy preserving manner. We show our experimental results in Section 6. The last section concludes the paper.

2. Related work

In data mining, several efforts have been made to preserve the privacy of individual records using randomization techniques [7,8,18,29,16] and to preserve the privacy of the database while running data mining algorithms over multiple data sources using cryptographic techniques such as secure multi-party computation and encryption [13,23,15]. Agrawal and Haritsa [8] proposed a privacy preserving data mining scheme using random perturbation. Vaidya and Clifton [33] studied privacy preserving distributed association rule mining over vertically partitioned data and Kantarcioglu and Clifton [22] studied the privacy preserving association rule mining over horizontally partitioned data using cryptographic tools such as secure multi-party computation and commutative encryption. In addition to these, Evfimievski et al. [18] proposed an approach to conduct association rule mining based on randomized response techniques. In particular, our work is closely related to [16,23,15], which are proposed to build decision trees in a privacy preserving manner. Du and Zhan [16] use randomization to build approximate decision trees. In their scheme, all data sources gather their data in a central database after perturbation and the central database runs the decision tree learning algorithm on the approximate data. Furthermore, Du and Zhan [15] propose a method to build a decision tree over vertically partitioned data using secure scalar product protocol. Lindell and Pinkas [23] propose a secure algorithm to build a decision tree using ID3 over horizontally partitioned data among two parties using secure multi-party computation. Although their technique can be generalized to more than two parties, it is inefficient and not scalable for a large number of parties [26]. Similar to these efforts, we propose a method to build decision trees over a large number of horizontally partitioned data. Our scheme is efficient and can scale up to thousands of private data sources.

Privacy preserving data management has been studied in the areas of databases and information retrieval. Recently, there has been a great interest in the database area for privacy preserving database operations such as intersection, join and aggregation operations. The goal is to answer queries over multiple databases without revealing any additional information other than the query result [3,17]. Agrawal et al. [3] use a commutative encryption to answer intersection and join queries over two private databases. Emekci et al. [17] answers the aggregation queries as well as intersection and join queries using Shamir's secret sharing with third parties. In this paper, we adapted the solution in [17] to be used in privacy preserving decision tree learning. The difference between this work and [17] is that we eliminate the need for third party and propose a new method without using third parties. Furthermore, a new method to check the correctness of results is introduced since the data sources can be malicious. Similar to [3,17], Aggarwal et al. [2] propose a method for finding the k th ranked element approximately in the union of more than two databases using secure multi-party computation.

In addition to these, there are several proposals in privacy preserving data outsourcing [20,21], which propose a new approach for data management, i.e., database as a service, and the goal is to protect the content of the data from database service provider while ensuring efficient query processing. Several high level design efforts and requirement specifications have been made to support the privacy of individual information while still supporting some degree of sharing [1,4–6,9,31,32]. On the other hand, in privacy preserving information retrieval, several methods has been introduced to preserve the privacy of the query poser by hiding the records retrieved from the data source [10,12,11,19].

3. Background

Data mining aims at extracting useful information out of a large collection of data. The rapid increase in the amount of available data in many fields, such as data streams, data-warehouses and bioinformatics, makes data mining an essential part of data management. Example data mining tasks include classification, clustering and association rule mining. We focus on classification problems. Given a set of possible categories and descriptions of objects (training set), the goal is to assign the correct category of the new objects. Classification has many applications in real world, such as stock planning of large superstores, medical diagnosis, etc.

3.1. Decision trees

A well known algorithm for classification is the decision tree algorithm. A decision tree is a tree whose internal nodes and leaf nodes are associated with object attributes and categories, respectively. Each internal node corresponds to a test on a single attribute and has outgoing edges that are the possible outcomes of the test (possible values of the associated attribute). Each leaf node is assigned a single category. In order to classify an object into a category, starting from the root of the decision tree, the object is tested for the corresponding attribute at each internal node and then moved down the tree along the edge corresponding to the object's value for that attribute. The traversal of the tree is continued until a leaf node is reached. Thus each object follows a single path from the root to a leaf of the decision tree depending on its attribute values and then assigned the category associated with the leaf node.

Fig. 1, adopted from [25], shows a simple example of a training data set and the resulting decision tree. In this example, the objects are **days** with four attributes related to the weather conditions (with possible values shown in parenthesis): outlook (*sunny, overcast, rain*), temperature (*cool, mild, hot*), humidity (*high, normal*), and windy (*true, false*). The goal here is to learn the weather conditions suitable for playing tennis using the training set given in Fig. 1a. The resulting decision tree that correctly classifies each object in the training set is shown in Fig. 1b.

3.2. ID3 algorithm

The ID3 algorithm constructs a decision tree in a top-down manner from a given set of samples. It starts at the root and determines the attribute, among all, that provides the best classification of the objects by itself.

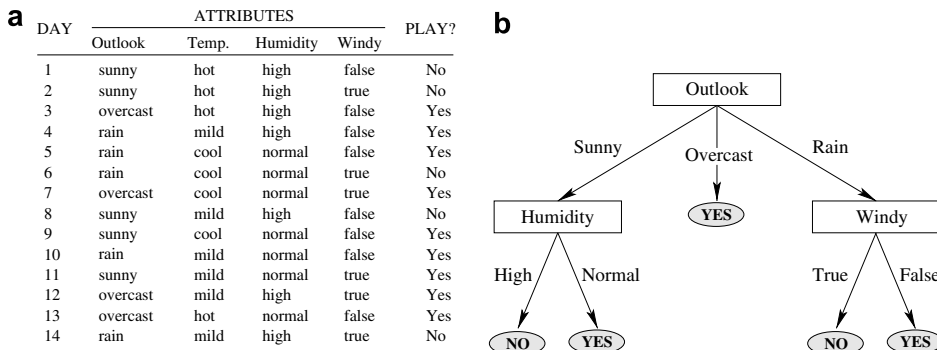


Fig. 1. Decision tree example. (a) Training set. (b) Decision tree.

This attribute is designated as the test attribute for the root and an outgoing edge is created for every possible value of the attribute. The new nodes are then constructed recursively in a similar manner by only considering the objects that have the corresponding values for the attributes associated with the nodes on the path from root to that node, and the attributes that are not selected yet. The tree generation along a path is terminated when all the attributes are used (the last node is assigned dominant category among the remaining objects) or all the remaining objects belong to the same category (the last node is assigned this category).

The attribute that gives the best prediction at each internal node is determined using information theory. The attribute that reduces the entropy of the category information the most is selected as the best attribute. Assume that there are k categories c_1, \dots, c_k , and a set T of objects whose categories are known. Let $T(c_i)$ be the set of objects with category c_i . Then the information needed to classify an object in T is:

$$E(T) = \sum_{i=1}^k \left(-\frac{|T(c_i)|}{|T|} \cdot \log \frac{|T(c_i)|}{|T|} \right)$$

Let us assume that the objects have n attributes A_1-A_n . In order to assess the prediction quality of an attribute A , we need to calculate the information needed to classify an object in T given its value for attribute A . Assume that A can have p values a_1-a_p . Then the information of T given A is:

$$E(T|A) = \sum_{i=1}^p \frac{|T(a_i)|}{|T|} \cdot E(T(a_i))$$

The information gain for each attribute A is:

$$\text{Gain}(A) = E(T) - E(T|A)$$

The best attribute A is then the one that has the maximum gain, i.e., minimum $E(T|A)$, among all considered attributes.

3.3. Distributed ID3

In our discussion of classification and ID3 so far we have assumed that the training data is available at a single location so that all necessary information such as the number of objects with a certain attribute value or category can be computed easily. One natural question that arises at this point is if the ID3 algorithm can be implemented efficiently in a setting where the training set is distributed over multiple sources instead of being stored at a single central location. The straightforward solution would of course be to have each data source send their data to a certain location (can be one of the data sources), compute the corresponding tree there over the entire data, and then send the resulting tree back to the data sources. However, this solution is not efficient since it requires the transfer of the entire data. Fortunately, it is possible to construct the tree with less communication cost. As explained in Section 3.2, the tree can be constructed if the distributed parties can compute $E(T|A)$ for every attribute A . Now, we will show how $E(T|A)$ can be computed for a given attribute A . Assume that there are k categories c_1 to c_k and A can have p values a_1-a_p . Let $T(a_i)$ be the set of objects whose A attribute value is a_i and $T(a_i, c_j)$ be the set of objects with value of A is a_i and category c_j .

$$\begin{aligned} E(T|A) &= \sum_{i=1}^p \frac{|T(a_i)|}{|T|} \cdot E(T(a_i)) = \frac{1}{|T|} \sum_{i=1}^p \left[|T(a_i)| \cdot \sum_{j=1}^k \left(-\frac{|T(a_i, c_j)|}{|T(a_i)|} \cdot \log \left(\frac{|T(a_i, c_j)|}{|T(a_i)|} \right) \right) \right] \\ &= \frac{1}{|T|} \left[-\sum_{i=1}^p \sum_{j=1}^k (|T(a_i, c_j)| \cdot \log(|T(a_i, c_j)|)) + \sum_{i=1}^p (|T(a_i)| \cdot \log(|T(a_i)|)) \right] \end{aligned} \tag{1}$$

Eq. (1) suggests that $E(T|A)$ value for an attribute A can be computed if all $|T(a_i)|$ and $|T(a_i, c_j)|$ values are known. Note that in this case, $|T(a_i)|$ and $|T(a_i, c_j)|$ should be computed over all data sources. If there are s data sources D_1 to D_s , then

$$|T(a_i)| = \sum_{l=1}^s |T_l(a_i)| \quad \text{and} \quad |T(a_i, c_j)| = \sum_{l=1}^s |T_l(a_i, c_j)|,$$

where $T(a_i)$ is the set of objects whose value for attribute A is a_i in D_i 's data set ($T(a_i, c_j)$ is also defined similarly).

Thus, the value of $E(T|A)$ can be calculated efficiently as follows. Data sources send their values for all $|T(a_i)|$ and $|T(a_i, c_j)|$ to a certain location, which then calculates the result using Eq. (1) and sends it back to data sources. Using the above algorithm for finding the gain for an attribute, the whole decision tree can be constructed as follows: Starting from the root node, the value of $E(T|A)$ is calculated for each attribute as described above and the attribute with the minimum value is assigned to the root node. Then each data source is notified of the result so that they partition their data accordingly. Thereafter the other nodes of the tree are constructed recursively in a similar manner.

3.4. Secure ID3 construction

In Lindell and Pinkas [24] have proposed a method for running approximate ID3 algorithm¹ over two parties in a privacy preserving manner such that no party learns any extra information other than the output of the algorithm, i.e., the resulting tree.

Since, the algorithm only needs to determine the name of the attribute that minimizes $E(T|A)$, they ignore $\frac{1}{|T|}$ and replace \log_2 with natural logarithm in Eq. (1). Thus, $E(T|A)$ can be computed as the sum of expressions of the form $(v1 + v2) \cdot \ln(v1 + v2)$ where $v1$ and $v2$ are the values known to the first party P_1 and second party P_2 , respectively (such as $v1 = |T_1(a_i)|$ and $v2 = |T_2(a_i)|$). Then they propose a method (called *private xlnx protocol*) for privately computing $x \cdot \ln x$ value. This protocol takes as input the values x_1 and x_2 from parties P_1 and P_2 , respectively and outputs random shares of an approximation of $x \cdot \ln x$ value where $x = x_1 + x_2$.

The attribute A with minimum $E(T|A)$ value is then privately determined through a two stage process: In the first stage, the parties use the private *xlnx* protocol to obtain random shares of the corresponding conditional entropy value for every attribute A . To compute these shares, both parties invoke the private *xlnx* protocol for every attribute, every possible value of that attribute and every category. In the second stage, the parties use Yao's two-party computation protocol [34] to determine the attribute with minimum conditional entropy. The protocol constructs a circuit that takes the random shares obtained by each party for each attribute in the first stage as input, computes the corresponding conditional value for each attribute, and then outputs the name of the attribute with minimum value.

As discussed in [26], this method can be generalized for multiple parties, however it is impractical to use it for a large number of parties due to high communication and computation costs involved.

4. Privacy preserving summation of multiple secrets

In this section, we will describe how to compute the summation of the secret values over n parties without revealing the secrets to other parties, where $n > 2$. We will first describe Shamir's secret sharing scheme in Section 4.1 and then explain how it can be used for privacy preserving summation of multiple secrets (Section 4.2). Finally, we propose an efficient technique to verify the correctness of the computation in Section 4.3.

4.1. Shamir's secret sharing

Shamir's secret sharing method [30] allows a dealer D to distribute a secret value v_s among n peers $\{P_1, P_2, \dots, P_n\}$, such that the knowledge of any k ($k \leq n$) peers is required to reconstruct the secret. Shamir's method is information theoretically secure since the complete knowledge of up to $k - 1$ peers does not reveal any information about the secret. Dealer D chooses a publicly known set X of n values and a hidden random polynomial $q(x)$ of degree $k - 1$, where the constant term of $q(x)$, i.e., $q(0)$, is the secret value v_s . The dealer then computes the share of each peer P_i as $q(X_i)$ and sends it to P_i . The method is summarized in Algorithm 1.

¹ Approximate ID3, ID3 _{δ} , includes a small modification to the ID3 algorithm such that instead of selecting the attribute that gives the maximum gain, any attribute whose gain is within δ -neighborhood of the maximum gain can be selected at each internal node during the construction of the tree, i.e., $\text{Gain}(A_{\text{selected}}) \geq (\text{Gain}(A_{\text{max_gain}}) - \delta)$ at each attribute selection step.

Algorithm 1. Shamir's secret sharing algorithm executed at D **Require:** v_s : Secret value, P : Set of parties P_1, \dots, P_n to distribute the shares, k : Number of shares required to reconstruct the secret.

- 1: Select a random polynomial $q(x) = a_{k-1}x^{k-1} + \dots + a_1x^1 + v_s$, where $a_{k-1} \neq 0$.
- 2: Choose n publicly known distinct random values x_1, \dots, x_n such that $x_i \neq 0$
- 3: Compute the share of each peer, P_i , where $share_i = q(x_i)$
- 4: **for** $i = 1$ to n **do**
- 5: Send $share_i$ to peer P_i
- 6: **end for**

In order to construct the secret value v_s , any set of k peers will need to share the information they have received. Using k shares, the polynomial $q(x)$, and thus the secret value v_s , can be determined. $q(x)$ can be reconstructed using Lagrange interpolation such that $p(x_i) = share_i$ where $i = 1, \dots, k$ (This is further explained in Section 4.2). The key observation here is that at least k shares are required to determine the random polynomial $q(x)$ of degree $k - 1$.

4.2. Privacy preserving summation

In this section, we show how Shamir's secret sharing algorithm can be used to privately compute the sum of secret values of multiple parties without revealing the secret values to others. The goal is to compute the sum $\sum_i v_i$, where v_i is the secret value of party P_i , without revealing any other information to parties. Thus at the end of the computation, each party P_i only knows its own value v_i (that it already knew) and the sum of the secret values of all parties involved, but nothing else.

A naive way to compute the sum of secret values of 4 parties P_1 – P_4 , i.e., $\sum_{i=1}^4 v_i$, such that parties would only learn the sum is as follows: P_1 chooses a random number r and sends $v_1 + r$ to P_2 , P_2 adds its value and sends $v_1 + v_2 + r$ to P_3 , and so on. Finally, P_4 sends $v_1 + v_2 + v_3 + v_4 + r$ back to P_1 , which then subtracts the random number r from the results and sends the value for the sum $\sum_{i=1}^4 v_i$ to all parties. This scheme, however, is not very secure since two parties could collude and learn the secret value of another party. For example, P_1 and P_3 could exchange the temporary result and learn the value of P_2 without revealing their values. A more sophisticated protocol to solve the problem could consist of two rounds. During the first round, every party P_i adds its value v_i and a random number r_i to the intermediate result. Therefore, P_1 sends $v_1 + r_1$ to P_2 , P_2 sends $v_1 + v_2 + r_1 + r_2$ to P_3 , and so on. P_1 then receives $\sum_{i=1}^4 (v_i + r_i)$ from P_4 . In the second round, every party starting from P_1 subtracts its random number r_i it added in the first round from the intermediate result and passes it to the next party. At the end of the second round, P_1 gets the sum $\sum_{i=1}^4 v_i$ and sends it to other parties. Although this scheme seems more secure than the first one, it still has some vulnerabilities. For example, P_2 and P_4 could learn the value of P_1 as follows. In the first round, P_4 would have the value $\sum_{i=1}^4 (v_i + r_i)$ and in the second round P_2 would get $v_1 + v_2 + r_2 + v_3 + r_3 + v_4 + r_4$ from P_1 , thus, P_2 and P_4 could figure out the random value r_1 of P_1 . Since, P_2 would also know $v_1 + r_1$ from the first round, they could deduce the secret value v_1 of P_1 .

In order to compute the sum of secret values in a privacy preserving manner, we use Shamir's secret sharing method. The main observation is that Shamir's secret sharing method allows one to compute any linear combination of secrets. We first outline the general solution and then further discuss its correctness and security aspects.

This technique computes the sum in three phases: *Distribution phase*, *Intermediate computation phase* and *Final computation phase*. In the distribution phase, n parties decide on the degree k of a polynomial that is going to be used in Shamir's secret sharing (The degree of the polynomial, k , should be less than or equal to $n - 1$). They also agree on $m \geq n$ random values $X = \{x_1, \dots, x_m\}$. For the sake of simplicity and without loss of generality, we will assume that the degree of the polynomial is $n - 1$ and $m = n$. Each party P_i has a secret value v_i and chose a random polynomial $q_i(x)$ of degree k and $q_i(0) = v_i$. P_i then creates n shares of its secret value, $sh(v_i, P_1), \dots, sh(v_i, P_n)$, using Shamir's secret sharing algorithm for each of the parties including itself. P_i computes the share of each party P_j as $sh(v_i, P_j) = q_i(x_j)$, using Algorithm 1. At the end of the distribution phase, every party P_i sends out the shares such that $sh(v_i, P_j)$ is sent to party P_j .

Example 1. Assume that there are 4 parties P_1 – P_4 with secret values $v_1 = 2$, $v_2 = 4$, $v_3 = 6$ and $v_4 = 8$ respectively and they want to compute $v_1 + v_2 + v_3 + v_4$ without revealing their values to each other. They decide on a polynomial degree $k = 3$ and $m = 4$ values $X = \{3, 5, 7, 8\}$. Each party P_i then chooses a random polynomial $q_i(x)$ of degree $k = 3$ whose constant term is the secret value v_i . P_1 picks $q_1(x) = x^3 - 2x^2 + 3x + 2$ and computes the shares for other parties such that the share of party P_j , $sh(v_1, P_j)$, is equal to $q_1(x_j)$, where x_j is the j th element of X . Thus the shares computed by P_1 are as follows: $sh(v_1, P_1) = q_1(3) = 20$, $sh(v_1, P_2) = q_1(5) = 92$, $sh(v_1, P_3) = q_1(7) = 267$, and $sh(v_1, P_4) = q_1(8) = 408$. Similarly, other parties P_2 , P_3 and P_4 pick random polynomials $q_2(x) = x^3 + x^2 - 6x + 4$, $q_3(x) = x^3 - 4x^2 - 3x + 6$, $q_4(x) = 2x^3 - x^2 - x + 8$, and compute the shares for other parties:

share(v_i, P_j)	P_1	P_2	P_3	P_4
v_2	$q_2(3)$	$q_2(5)$	$q_2(7)$	$q_2(8)$
v_3	$q_3(3)$	$q_3(5)$	$q_3(7)$	$q_3(8)$
v_4	$q_4(3)$	$q_4(5)$	$q_4(7)$	$q_4(8)$

During the intermediate computation phase, each party adds up all the shares it received from other parties and then sends this intermediate result to all other parties. Party P_i gets the shares $q_1(x_i), q_2(x_i), \dots, q_n(x_i)$ and sends the intermediate result, $INTERRES_i = q_1(x_i) + q_2(x_i) + \dots + q_n(x_i)$ to all parties.

In the final computation phase, each party P_i can compute the sum of secret values using the intermediate results it received during the previous phase. Note that each party P_i uses a random polynomial with degree $k = n - 1 = 3$ and constant term v_i , and therefore the sum of all these polynomials results in another polynomial $S(x) = q_1(x) + q_2(x) + \dots + q_n(x)$ of degree 3 with the constant term equal to the sum of all secret values, $v_1 + v_2 + \dots + v_n$. The intermediate results computed during intermediate computation phase corresponds to the values of $S(x)$ at n points, x_1 – x_n . At the beginning of the final computation phase, each party P_i will have the n values of polynomial $S(x)$ of degree $n - 1$ and thus it can determine all the coefficients of $S(x)$. Since the constant term of $S(x)$ is equal to $\sum_{i=1}^n v_i$, each party will learn the sum of all values, but none of the individual ones. In **Example 1**, the sum polynomial $S(x) = \sum_{i=1}^4 q_i(x)$ is equal to $5x^3 - 7x^2 - 7x + 20$, where 20 is the sum of the secret values $2 + 4 + 6 + 8$. In addition, $S(x_i) = \sum_{j=1}^4 q_j(x_i)$ for all $i \leq n = 4$. Hence each party can determine the coefficients of $S(x)$ (since it is of degree 3 and its value at 4 points are known) and thus the sum of all the secret values.

4.2.1. Summary and proof of correctness

In the above algorithm, each party P_j selects a random polynomial of the form $a_{P_j}x_i^{n-1} + b_{P_j}x_i^{n-2} + \dots + v_{P_j}$. After generating its random polynomial, P_j sends other parties their shares where the share of each party P_i is computed as $sh(v_j, P_i) = a_{P_j}x_i^{n-1} + b_{P_j}x_i^{n-2} + \dots + v_{P_j}$. After a party P_i receives its share from all other parties, it computes its intermediate result by summing its shares:

$$\begin{aligned}
 & a_{P_1}x_i^{n-1} + b_{P_1}x_i^{n-2} \dots + v_{P_1} + \\
 & a_{P_2}x_i^{n-1} + b_{P_2}x_i^{n-2} \dots + v_{P_2} + \\
 & \vdots \\
 & a_{P_n}x_i^{n-1} + b_{P_n}x_i^{n-2} \dots + v_{P_n}
 \end{aligned}$$

P_i then sends its intermediate result $INTERRES_i = (a_{P_1} + a_{P_2} + \dots + a_{P_n})x_i^{n-1} + \dots + SUM$, where SUM is the sum of secret values ($SUM = v_{P_1} + v_{P_2} + \dots + v_{P_n}$). Each party receives n results from every other party and has n intermediate results:

$$\begin{aligned}
 INTERRES_1 &= (a_{P_1} + a_{P_2} + \dots + a_{P_n})x_1^{n-1} + \dots + SUM \\
 INTERRES_2 &= (a_{P_1} + a_{P_2} + \dots + a_{P_n})x_2^{n-1} + \dots + SUM \\
 &\vdots \\
 INTERRES_n &= (a_{P_1} + a_{P_2} + \dots + a_{P_n})x_n^{n-1} + \dots + SUM
 \end{aligned}$$

Since there are n unknown coefficients (including SUM) and n equations, each party P_i can solve the above set of equations and determine the value of $SUM = \sum_{i=1}^n v_{P_i}$, however it cannot determine the secret values of the other parties since the individual polynomial coefficients selected by other parties are not known to P_i . The whole process is summarized in Algorithm 2.

Algorithm 2. Privacy preserving summation of secrets, executed at each party P_i

Require: P : Set of parties P_1, \dots, P_n ,
 v_i : Secret value of P_i ,
 X : A set of n publicly known random values x_1, \dots, x_n such that $x_i \neq 0$,
 k : Degree of the random polynomial

- 1: Select a random polynomial $q_i(x) = a_{k-1}x^{k-1} + \dots + a_1x^1 + v_i$
- 2: /* Distribution phase */
- 3: Compute the share of each party, P_j , where $sh(v_i, P_j) = q_i(x_j)$
- 4: **for** $j = 1$ to n **do**
- 5: Send $sh(v_i, P_j)$ to peer P_j
- 6: **end for**
- 7: /* Intermediate computation phase */
- 8: Receive the shares $sh(v_j, P_i)$ from every party P_j
- 9: Compute $INTERRES_i = \sum_{j=0}^n sh(v_j, P_i)$
- 10: **for** $j = 1$ to n **do**
- 11: Send $INTERRES_i$ to peer P_j
- 12: **end for**
- 13: /* Final computation phase */
- 14: Receive the intermediate results $INTERRES_j$ from every party P_j
- 15: Solve the set of equations to find the $SUM = \sum_{j=0}^n v_j$

In this method, the parties cannot learn the secret value of another party even if they exchange their shares with each other (stated in Lemma 1). The secret value v_i of P_i can only be revealed if all the remaining $n - 1$ parties collude and subtract the sum of their values from the overall sum, however there is no way to avoid this problem since the algorithm is expected to output the sum of all secret values to all parties. Note that this scenario is very unlikely considering that a large number of parties will be participating in the computation. Furthermore, a malicious party may not be willing to participate in such a plot, because then the remaining parties might also collude to learn its own value.

Lemma 1. *The secret value v_i of a party P_i cannot be revealed even if all the remaining parties exchange their shares.*²

Proof. Since each party P_i executes Shamir's secret sharing algorithm with a random polynomial of degree $n - 1$, the value of that polynomial at n different points are needed in order to compute the coefficients of the corresponding polynomial, i.e., the secret value of party P_i . P_i computes the value of its polynomial at n points as shares, and then keeps one of these shares for itself and sends the remaining $n - 1$ shares to other parties. Since all n shares are needed to reveal the secret in Shamir's secret sharing method, other parties cannot compute v_i even if they combine their shares. \square

Additionally, an external adversary listening to the network traffic of the parties will not be able to obtain any useful information if it does not know the selected $X = x_1, x_2, \dots, x_n$ values (see Lemma 2). Therefore, the parties can exchange the X values through some secure means of communication and then safely execute the protocol over arbitrary, not necessarily secure, channels.

² Note that if the remaining $n - 1$ parties collude, they cannot reveal v_i by combining their shares, but they can do so by subtracting the sum of their secret values from the overall sum.

Lemma 2. *An external adversary listening to the network traffic of the parties cannot learn any useful information, such as the secret values or the sum of those values.*

Proof. An adversary listening to the network traffic of the parties will be able to obtain all the shares and the intermediate values, however these values cannot be used to determine the coefficients of the sum polynomial without knowing the x values for which the intermediate results are calculated. \square

4.3. Computation with malicious parties

The technique proposed in Section 4.2 assumes that all parties do the computations honestly in the distribution and intermediate computation phase. In cryptography, there are several methods to verify the correctness of the shares distributed by a dealer [28]. In our case, the dealer is the data source itself. Hence the dishonest computation of party P_i during the distribution phase, i.e., reporting of incorrect share values, means that P_i is not willing to share its correct value with others. Since a party can always change its own data, it is impossible to detect such kind of malicious behavior. Therefore, we will assume all parties are honest during the distribution phase. However, a party P_i may start behaving dishonestly during the intermediate computation phase. P_i can send an incorrect intermediate result to other parties so that they will not be able to compute the correct final result. On the other hand, P_i can still compute the correct result since it knows the correct value of its intermediate result. In this section, we propose a computation efficient technique to verify the correctness of the final result assuming that the parties compute the shares honestly but they might act maliciously during the intermediate computation phase.

In the original technique, at the final computation phase every party will have n intermediate results from which it can compute the final sum value. Even if one of the intermediate results is not correct, then the final result computed will be incorrect and the parties will not be able to detect this. The observation here is that if there are more than n equations with n unknowns, then a party can choose any n of these intermediate results to solve the equation set. Furthermore, it verifies the correctness of the final result by solving different sets of n equations.

To implement this scheme for verifying the final result, the parties can use random polynomials of degree $2n - 2$ instead of $n - 1$. In this case, the parties agree on a set of $2n$ points, $X = \{x_1, x_2, \dots, x_{2n}\}$, and each party P_i selects a random polynomial $q_i(x)$ with degree $2n - 2$ and constant coefficient v_i . Now each party P_i computes 2 shares for each of the n parties, such that the shares for party P_j are $sh_1(v_i, P_j) = q_i(x_{2j-1})$ and $sh_2(v_i, P_j) = q_i(x_{2j})$. After a party receives all $2n$ of its shares, it computes the intermediate result for each of its shares separately and sends both intermediate results to other parties. As a result, each party will have $2n$ equations at the final computation phase in order to compute $2n - 1$ unknowns, one of which is the final result. The unknowns can be solved by selecting any $2n - 1$ intermediate results from the available $2n$ results.

This selection can be made in $\binom{2n}{2n-1} = 2n$ different ways. Thus, the parties can now compare the final result of different solutions to ensure the correctness. If all parties behave honestly, then all possible equation sets produce the same value as the final result. Note that both Lemma 1 and Lemma 2 still hold for this scheme (the proofs are similar) so that:

- The secret value v_i of a party P_i cannot be revealed even if all other parties exchange their shares.
- An external adversary listening to the network traffic of the parties cannot learn any useful information, such as the secret values or the sum of those values.

The above technique might be computationally intensive for large n because in the worst case all $2n$ different sets of equations need to be solved. We try to optimize this when at most t of n parties can be malicious, and give a probabilistic guarantee on the correctness of the final result. Our optimization is based on comparing the results of any 2 equation sets rather than comparing all $2n$ possible equation sets. If the results of the two equations sets are not equal, then there is at least one malicious party and the final result is incorrect. However, if they are equal, we cannot say the final value is correct but we can give probabilistic guarantees on the correctness.

We assume that the parties will not be able to know the intermediate results of other parties before sending out their intermediate results. We will look at two different scenarios: (1) *Individual attack* where malicious parties do not collude, (2) *Collaborative attack* where all t malicious parties collude. For individual attacks, in order to get the same final result from two different sets, the malicious party needs to guess the intermediate results of other parties correctly. Thus the probability of obtaining the same final result for two different equation sets is $1/|\text{domain}|^{2n-2}$, where *domain* is the possible values that the sum polynomial can take. Since, the malicious parties do not collude, this probability does not change with the number of malicious parties. In the case of collaborative attack, since t malicious parties know their own intermediate results, the probability of obtaining the same final result for two different equation sets is $1/|\text{domain}|^{2n-2t}$. Since *domain* is usually arbitrarily large, above probabilities are very small, and thus the parties can be confident about the correctness of the final result they computed.

The above scheme assumes that the parties cannot know the intermediate results of other parties before sending their intermediate results. In order to satisfy this assumption, the parties can change their intermediate results in two rounds. In the first round, each party sends its intermediate results to other parties in an encrypted format. After receiving all the encrypted intermediate results, it sends the corresponding decryption key to other parties in the second round.

5. Privacy preserving ID3

In Section 4.2, we showed how to compute the summation of secret values available at multiple parties without revealing the actual values to other parties. Algorithm 3 shows how this could be used to implement the ID3 algorithm over multiple data sources in a privacy preserving manner.

Algorithm 3. Privacy preserving construction of a decision tree over multiple parties using ID3 algorithm

Require: R : Set of attributes to be considered,

O : Set of objects to be considered,

$C = \{c_1, c_2, \dots, c_k\}$: Set of possible categories.

1: **if** R is empty **then**

2: Return a leaf node whose category is set to the dominant category among the objects in O

/* For each category c_i , parties compute the number of objects with category c_i using the private summation protocol described in Section 4.2 (each party uses the number of objects with category c_i in its data set as its secret value). Then the category with the maximum sum value is selected. */

3: **end if**

4: **if** All objects in O have the same category c_i **then**

5: Return a leaf node whose category is set to c_i

/* Using the total number of objects within each category calculated in the previous case, parties can check if all but a single category c_i still has objects. */

6: **end if**

7: Determine the attribute A that best classifies the objects in O and assign it as the test attribute for the current tree node

/* For every attribute A in R , the parties compute $|T(a_i)|$ and $|T(a_i, c_j)|$ for every possible value a_i of A , and for every possible category c_j , using the private summation protocol. The conditional entropy value $E(T|A)$ for each attribute A is obtained using Eq. (1), and the attribute with the minimum $E(T|A)$ value (i.e., the highest gain) is selected as the best attribute. */

8: Create a new node for every possible value a_i of A , and recursively call this method on it with

$R' = (R - \{A\})$ and $O' = O(a_i)$ /*

Each party knows the attribute selected and the attribute values, and thus partitions its local data set accordingly. */

The communication complexity for each node in the above algorithm (mainly step 7) is as follows. As seen in Section 3.3, the private summation protocol is executed $p(k+1)$ times for each attribute, where p is the pos-

sible values of the attribute and k is the number of categories. Since there are $|R|$ attributes to be considered, the private summation protocol is executed $O(p \cdot k \cdot |R|)$ times. For the private summation protocol, each party sends and receives n shares in the distribution phase, and then sends and receives n intermediate results in the intermediate result computation phase. Thus the communication cost for private summation protocol at each party is $O(n)$. We also measured the time it takes for Matlab to compute the final result from a set of 1000 intermediate results (this corresponds to the multiplication of two matrices with sizes 1000×1000 and 1000×1) and it took approximately 2.5 s.

Note that this protocol allows all the parties to learn the conditional entropy values of each attribute while determining the best attribute at a node. Additionally, as a result of the computations at step 2 of Algorithm 3, a party can learn that no other party has any objects with a certain category c_i if the total number of objects with category c_i computed is equal to the number of such objects it has in its data set. We believe that these types of information leakages are acceptable considering the efficiency and scalability properties offered by our protocol.

6. Experimental results

In this section, we will present experimental results. In our experiments, we showed how collaboration reduces classification error and how our system scales in terms of the number of parties collaborating. We used two datasets from [14]:

- **Mfeat-feature Dataset:** The dataset consists of features of handwritten numerals extracted from a collection of Dutch utility maps. It contains 10 classes and 649 attributes. 200 patterns per class have been digitized in binary images (for a total of 2000 patterns), where each pattern instance is in one of the 10 classes.
- **Nursery Dataset:** This dataset was derived from a hierarchical decision model originally developed to rank applications for nursery schools in Ljubljana, Slovenia in 1980s. The dataset contains a total of 12,960 instances and 8 attributes, where each instance belongs to one of the 5 classes.

In our experiments, we use 2/3 of the datasets for training and use the remaining for measuring the accuracy of classification. We distribute the datasets to 128 parties such that the distribution of instances in parties are similar. After that, we build a decision tree over the union of k parties' data by varying k from 1 to 128, and then measure the accuracy of the resulting decision tree. Fig. 2 shows the accuracy of the resulting decision tree for different values of k . The accuracy of the decision tree increases if the number of parties involved increases. If parties build their decision trees by themselves without collaborating with other parties, the accuracy of classification is 40.6% in Mfeat dataset and 71.3% in Nursery dataset. The reason of low accuracy is that we need more samples to avoid over-fitting. The accuracy for the Mfeat dataset is less than the accuracy for Nursery dataset, since the number of instances is less and the number of attributes and classes are more in

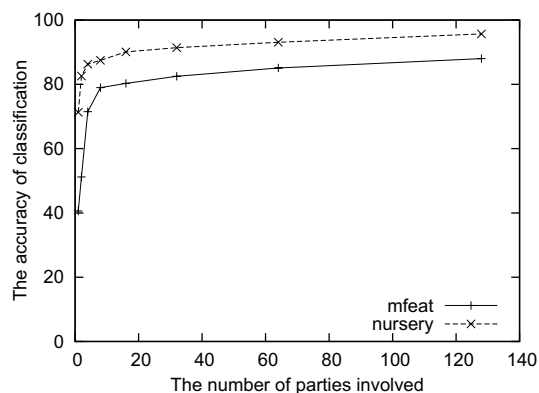


Fig. 2. The accuracy of the decision tree with different number of parties collaborating.

Table 1
Running time for Mfeat dataset

The number of parties	Running time (in s)	Running time (in h)
1	10.80	0.0030
2	43.20	0.0122
4	172.82	0.0480
8	691.28	0.1920
16	2765.11	0.7680
32	11060.37	3.0723
64	44241.36	12.2892
128	176965.17	49.1569

Table 2
Running time for nursery dataset

The number of parties	Running time (in s)	Running time (in h)
1	8.00E-4	2.22E-7
2	0.0032	8.89E-7
4	0.012	3.55E-6
8	0.051	1.42E-5
16	0.204	5.67E-5
32	0.819	2.27E-4
64	3.277	9.10E-4
128	13.10	0.0036

Mfeat dataset. If all parties build decision tree over the union of their data the accuracy of the final decision tree is 88% in Mfeat dataset and 95.7% in Nursery dataset. In order to get an accuracy over 80%, at least 16 of the parties should collaborate and build the decision tree together in a privacy preserving manner. These results demonstrate that in order to build accurate decision trees we need more training data and thus parties should build decision trees collaboratively.

In addition to accuracy, we also measure how well our approach scales for large number parties in terms of running time. Tables 1 and 2 show the running time of the proposed algorithms for different number of parties for Mfeat and Nursery datasets, respectively. In both cases, the running time is reasonable and scales well with the increasing number of parties. The running time over Mfeat dataset is more than the running time over Nursery dataset since it has more attributes, more possible values for each attribute, and more classes (we need more rounds of secure summation).

7. Conclusions

In this paper, we propose a novel method to build a decision tree over an arbitrary number of data sources using the well known ID3 algorithm. Furthermore, we propose an efficient method to verify the correctness of the final results even when a large number of parties are involved. Our method uses communication and computation efficient techniques to construct the decision tree in a privacy preserving manner and thus it can scale up to thousands of data sources.

Acknowledgement

This research is supported by the NSF Grants under CNF 04-23336, and IIS 02-23022.

References

- [1] Gagan Aggarwal, Mayank Bawa, Prasanna Ganesan, Hector Garcia-Molina, Krishnamurthy Kenthapadi, Nina Mishra, Rajeev Motwani, Utkarsh Srivastava, Dilys Thomas, Jennifer Widom, Ying Xu, Enabling privacy for the paranoids, in: VLDB, 2004, pp. 708–719.

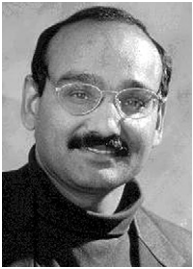
- [2] Gagan Aggarwal, Nina Mishra, and Benny Pinkas, Privacy-preserving computation of the k 'th-ranked element, in: IACR Eurocrypt, 2004 pp. 40–55.
- [3] R. Agrawal, A. Evfimievski, R. Srikant, Information sharing across private databases, in: SIGMOD, 2003, pp. 86–97.
- [4] Rakesh Agrawal, Peter J. Haas, Jerry Kiernan, A system for watermarking relational databases, in: SIGMOD, 2003, pp. 674–674.
- [5] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, Yirong Xu, Hippocratic databases, in: VLDB, 2002.
- [6] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, Yirong Xu, Implementing p3p using database technology, in: ICDE, 2003.
- [7] Rakesh Agrawal, Ramakrishnan Srikant, Privacy-preserving data mining, in: SIGMOD, 2000, pp. 439–450.
- [8] Shipra Agrawal, Jayant R. Haritsa, A framework for high-accuracy privacy-preserving mining, in: ICDE, 2005.
- [9] Mayank Bawa, Roberto Bayardo Jr., Rakesh Agrawal, Privacy preserving indexing of documents on the network, in: VLDB, Aug 2003, pp. 922–933.
- [10] Christian Cachin, Silvio Micali, Markus Stadler, Computationally private information retrieval with polylogarithmic communication, Lecture Notes in Computer Science 1592 (1999) 402–414.
- [11] B. Chor, O. Goldreich, E. Kushilevitz, M. Sudan, Private information retrieval, in: FOCS, 1995, pp. 41.
- [12] Benny Chor, Niv Gilboa, Computationally private information retrieval (extended abstract), in: STOC, 1997, pp 304–313.
- [13] Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, Michael Y. Zhu, Tools for privacy preserving distributed data mining, SIGKDD Explor. Newsl. 4 (2) (2002) 28–34.
- [14] C.L. Blake, D.J. Newman, S. Hettich, C.J. Merz, UCI Repository of Machine Learning Databases (1998).
- [15] Wenliang Du, Zhijun Zhan, Building decision tree classifier on private data, in: CRPITS, 2002, pp. 1–8.
- [16] Wenliang Du, Zhijun Zhan, Using randomized response techniques for privacy-preserving data mining, in: KDD, 2003, pp. 505–510.
- [17] Fatih Emekci, Divyakant Agrawal, Amr El Abbadi, Aziz Gulbeden, Privacy preserving query processing using p2p overlay networks, in: IEEE International Conference on Data Engineering (ICDE), 2006.
- [18] Alexandre Evfimievski, Ramakrishnan Srikant, Rakesh Agrawal, Johannes Gehrke, Privacy preserving mining of association rules, in: SIGKDD, 2002, pp. 217–228.
- [19] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, Tal Malkin, Protecting data privacy in private information retrieval schemes, in: STOC, 1998, pp. 151–160.
- [20] H. Hacigumus, B.R. Iyer, C. Li, S. Mehrotra, Executing SQL over encrypted data in the database service provider model, in: SIGMOD, 2002.
- [21] Bijit Hore, Sharad Mehrotra, Gene Tsudik, A privacy-preserving index for range queries, in: VLDB, 2004, pp. 720–731.
- [22] Murat Kantarcioglu, Chris Clifton, Privacy-preserving distributed mining of association rules on horizontally partitioned data, IEEE Transactions on Knowledge and Data Engineering 16 (9) (2004) 1026–1037.
- [23] Yehuda Lindell, Benny Pinkas, Privacy preserving data mining, in: Proc. of the 20th Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag, 2000, pp. 36–54.
- [24] Yehuda Lindell, Benny Pinkas, Privacy preserving data mining, Journal of Cryptology 15 (3) (2002) 177–206.
- [25] Tom M. Mitchell, Machine Learning, McGraw-Hill, New York, 1997.
- [26] Benny Pinkas, Cryptographic techniques for privacy-preserving data mining, SIGKDD Explorations (2003).
- [27] J.R. Quinlan, Induction of decision trees, in: Jude W. Shavlik, Thomas G. Dietterich, (Eds.), Readings in Machine Learning. Morgan Kaufmann, 1990. Originally published in Machine Learning, vol.1, 1986, pp.81–106.
- [28] T. Rabin, M. Ben-Or, Verifiable secret sharing and multiparty protocols with honest majority, in: STOC, 1989, pp. 73–85.
- [29] Shariq Rizvi, Jayant R. Haritsa, Maintaining data privacy in association rule mining, in: VLDB, Aug 2002, pp. 682–693.
- [30] A. Shamir, How to share a secret, Communications of the ACM 22 (11) (1979) 612–613.
- [31] Radu Sion, Mikhail Atallah, Sunil Prabhakar, Rights protection for relational data, in: SIGMOD, 2003, pp. 98–109.
- [32] Radu Siona, Mikhail J. Atallah, Sunil Prabhakar, Resilient rights protection for sensor streams, in: VLDB, 2004, pp. 876–887.
- [33] Jaideep Vaidya, Chris Clifton, Privacy preserving association rule mining in vertically partitioned data, in: KDD, 2002, pp. 639–644.
- [34] A.C. Yao, How to generate and exchange secrets, in: FOCS, 1986, pp. 162–167.



Dr. Fatih Emekci got his MS and PhD from the Department of Computer Science at the University of California, Santa Barbara in 2006 and is currently affiliated with Oracle Corporation. His research interests are in the areas of data warehousing, data privacy and query optimization. Dr. Emekci received a BS in Computer Science from Bilkent University, Ankara, Turkey in 2002.



Ozgur D. Sahin received his PhD and MS degrees in computer science from University of California, Santa Barbara in 2005, and BS in computer engineering from Bilkent University (Turkey) in 2001. His research interests include database systems and applications, distributed systems, peer-to-peer systems, and information retrieval. He is currently affiliated with Google Inc.



Professor Divyakant Agrawal has been on the faculty of the Department of Computer Science at the University of California, Santa Barbara, since 1987. His research interests are in the areas of distributed systems and databases. Professor Agrawal received a BE (Hons) in Electrical Engineering from the Birla Institute of Technology and Science (India) in 1980, and an MS and PhD in Computer Science from the State University of New York at Stony Brook in 1984 and 1987, respectively.



Professor Amr El Abbadi joined the Department of Computer Science at the University of California, Santa Barbara in August 1987. His research interests are in the areas of distributed algorithms, multidimensional and spatial databases and large-scale information systems. Professor El Abbadi received his BEng from Alexandria University (Egypt) in 1980 and his PhD in Computer Science from Cornell University in 1987.