Building Decision Tree Classifier on Private Data [*]

Wenliang Du   **Zhijun Zhan**

Center for Systems Assurance
Department of Electrical Engineering and Computer Science
Syracuse University, Syracuse, NY 13244,
Email: `wedu,zhzhan@ecs.syr.edu`

Abstract

This paper studies how to build a decision tree classifier under the following scenario: a database is vertically partitioned into two pieces, with one piece owned by Alice and the other piece owned by Bob. Alice and Bob want to build a decision tree classifier based on such a database, but due to the privacy constraints, neither of them wants to disclose their private pieces to the other party or to any third party.

We present a protocol that allows Alice and Bob to conduct such a classifier building without having to compromise their privacy. Our protocol uses an untrusted third-party server, and is built upon a useful building block, the scalar product protocol. Our solution to the scalar product protocol is more efficient than any existing solutions.

Keywords: Privacy, decision tree, classification.

1   Introduction

Success in many endeavors is no longer the result of an individual toiling in isolation; rather success is achieved through collaboration, team efforts, or partnerships. In the modern world, collaboration is important because of the mutual benefit it brings to the parties involved. Sometimes, such collaboration even occurs between competitors, mutually untrusted parties, or between parties that have conflicts of interests, but all parties are aware that the benefit brought by such collaboration will give them the advantage over others. For this kind of collaboration, data privacy becomes extremely important: all parties of the collaboration promise to provide their private data to the collaboration, but none of them wants the others or any third party to learn much about their private data.

Currently, to solve the above problem, a commonly adopted strategy is to assume the existence of a *trusted* third party. In today's dynamic and sometimes malicious environment, making such an assumption can be difficult or in fact not feasible. Solutions that do not assume the third party are greatly desired.

In this paper, we study a very specific collaboration, the data mining collaboration: two parties, Alice and Bob, each having a private data set, want to conduct data mining on the joint data set that is the union of all individual data sets; however, because of the privacy constraints, no party wants to disclose its private data set to each other or any third party. The objective of this research is to develop efficient methods that enable this type of computation while minimizing the amount of the private information each party has to disclose to the other. The solution to this problem can be used in situations where data privacy is a major concern.

Data mining includes various algorithms such as classification, association rule mining, and clustering. In this paper, we focus on classification. There are two types of classification between two collaborative parties: Figure 1.a (suppose each row represents a record in the data set) shows the data classification on the horizontally partitioned data, and Figure 1.b shows the data classification on the vertically partitioned data. To use the existing data classification algorithms, on the horizontally partitioned data, both parties needs to exchange some information, but they do not necessarily need to exchange each single record of their data sets. However, for the vertically partitioned data, the situation is different. A direct use of the existing data classification algorithms requires one party to send its data (every record) to the other party, or both parties send their data to a trusted central place (such as a super-computing center) to conduct the computation. In situations where the data records contain private information, such a practice will not be acceptable.

In this paper, we study the classification on the vertically partitioned data, in particular, we study how to build a decision tree classifier on private (vertically partitioned) data. In this problem, each single record is dividied into two pieces, with Alice knowing one piece and Bob knowing the other piece. We have developed a method that allows them to build a decision tree classifier based on their joint data.

2   Related Work

Privacy Preserving Data Mining.   In early work on such privacy preserving data mining problem Lindell and Pinkas (Lindell & Pinkas 2000) propose a solution to the privacy preserving classification problem using the oblivious transfer protocol, a powerful tool developed by the secure multi-party computation studies. The solution, however, only deals with the horizontally partitioned data, and targets only for the ID3 algorithm (because it only emulates the computation of the ID3 algorithm). Another approach for solving the privacy preserving classification problem was proposed by Agrawal and Srikant (Agrawal & Srikant 2000) and also studied in (Agrawal &
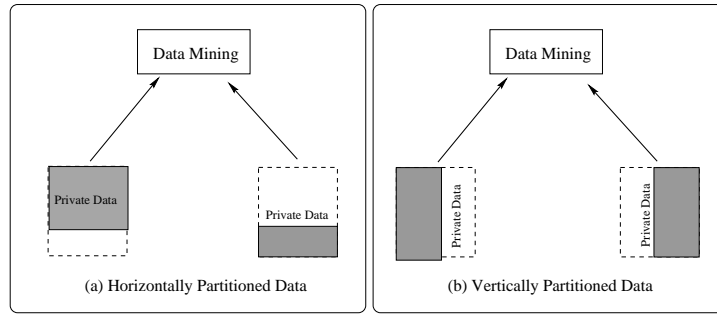

Figure 1: Privacy Preserving Data Mining

Aggarwal 2001). In this approach, each individual data item is perturbed and the distributions of the all data is reconstructed at an aggregate level. The technique works for those data mining algorithms that use the probability distributions rather than individual records. An example of classification algorithm which uses such aggregate information is also discussed in (Agrawal & Srikant 2000).

There has been research considering preserving privacy for other type of data mining. For instance, Vaidya and Clifton proposed a solution (Vaidya & Clifton 2002) to the privacy preserving distributed association mining problem.

Secure Multi-party Computation. The problem we are studying is actually a special case of a more general problem, the Secure Multi-party Computation (SMC) problem. Briefly, a SMC problem deals with computing any function on any input, in a distributed network where each participant holds one of the inputs, while ensuring that no more information is revealed to a participant in the computation than can be inferred from that participant's input and output (Goldwasser 1997). The SMC problem literature is extensive, having been introduced by Yao (Yao 1982) and expanded by Goldreich, Micali, and Wigderson (Goldreich, Micali & Wigderson 1987) and others (Franklin, Galil & Yung 1992). It has been proved that for any function, there is a secure multi-party computation solution (Goldreich 1998). The approach used is as follows: the function F to be computed is first represented as a combinatorial circuit, and then the parties run a short protocol for every gate in the circuit. Every participant gets corresponding shares of the input wires and the output wires for every gate. This approach, though appealing in its generality and simplicity, means that the size of the protocol depends on the size of the circuit, which depends on the size of the input. This is highly inefficient for large inputs, as in data mining. It has been well accepted that for special cases of computations, special solutions should be developed for efficiency reasons.

3 Decision-Tree Classification Over Private Data

3.1 Background

Classification is an important problem in the field of data mining. In classification, we are given a set of example records, called the training data set, with each record consisting of several attributes. One of the categorical attributes, called the class label, indicates the class to which each record belongs. The objective of classification is to use the training data set to build a model of the class label such that it can be used to classify new data whose class labels are unknown.

Many types of models have been built for classification, such as neural networks, statistical models, genetic models, and decision tree models. The decision tree models are found to be the most useful in the domain of data mining since they obtain reasonable accuracy and they are relatively inexpensive to compute. Most decision-tree classifiers (e.g. CART and C4.5) perform classification in two phases: *Tree Building* and *Tree Pruning*. In tree building, the decision tree model is built by recursively splitting the training data set based on a locally optimal criterion until all or most of the records belonging to each of the partitions bear the same class label. To improve generalization of a decision tree, tree pruning is used to prune the leaves and branches responsible for classification of single or very few data vectors.

3.2 Problem Definition

We consider the scenario where two parties, Alice and Bob, each having a private data set (denoted by S_a and S_b respectively), want to collaboratively conduct decision tree classification on the union of their data sets. Because they are concerned about the data's privacy, neither party is willing to disclose its raw data set to the other. Without loss of generality, we make the following assumptions on the data sets S_a and S_b (the assumptions can be achieved by pre-processing the data sets S_a and S_b, and such pre-processing does not require one party to send its data set to the other party):

1. S_a and S_b contain the same number of data records. Let N represent the total number of data records.
2. S_a contains some attributes for all records, S_b contains the other attributes. Let n represent the total number of attributes.
3. Both parties share the class labels of all the records and also the names of all the attributes.

Problem 3.1 (Privacy-preserving Classification Problem) Alice has a private training data set S_a and Bob has a private training data set S_b, data set $[S_a \cup S_b]$ is the union of S_a and S_b (by vertically putting S_a and S_b together so that the concatenation of the ith record in S_a with the ith record in S_b becomes the ith record in $[S_a \cup S_b]$). Alice and Bob want to conduct the classification on $[S_a \cup S_b]$ and finally build a decision tree model which can classify the new data not from the training data set.

3.3 Tree Building

3.3.1 The Schemes Used To Obtain The Information Gain

There are two main operations during tree building: (1) evaluation of splits for each attribute and selection of the best split and (2) creation of partitions using the best split. Having determined the overall best split, partitions can be created by a simple application of the splitting criterion to the data. The complexity lies in determining the best split for each attribute.

A splitting index is used to evaluate the "goodness" of the alternative splits for an attribute. Several splitting schemes have been proposed in the past (Rastogi & Shim 2000). We consider two common schemes: the *entropy* and the *gini* index . If a data set S contains examples from m classes, the $Entropy(S)$ and the $Gini(S)$ are defined as followings:

$$Entropy(S) = -\sum_{j=1}^{m} P_j \log P_j \qquad (1)$$

$$Gini(S) = 1 - \sum_{j=1}^{m} P_j^2 \qquad (2)$$

where P_j is the relative frequency of class j in S. Based on the entropy or the gini index, we can compute the information gain if attribute A is used to partition the data set S:

$$Gain(S, A) = Entropy(S) - \sum_{v \in A}(\frac{|S_v|}{|S|} * Entropy(S_v)) \qquad (3)$$

$$Gain(S, A) = Gini(S) - \sum_{v \in A}(\frac{|S_v|}{|S|} * Gini(S_v)) \qquad (4)$$

where v represents any possible values of attribute A; S_v is the subset of S for which attribute A has value v; $|S_v|$ is the number of elements in S_v; $|S|$ is the number of elements in S.

3.3.2 Terminology and Notation

For convenience, we define the following notations:

1. Let S_a represent Alice's data set; and let S_b represent Bob's data set.
2. We say node $\hat{A}$ is well classified if $\hat{A}$ contains records that belong to the same class; we say $\hat{A}$ is not well classified if $\hat{A}$ contains records that do not belong to the same class.
3. Let B represent an array that contains all of Alice and Bob's attributes and $B[i]$ to represent the ith attribute in B.

3.3.3 Decision Tree Building Procedure

The following is the procedure for building a decision tree on (S_a, S_b).

1. Alice computes information gain for each attribute of S_a. Bob computes information gain for each attribute of S_b. Initialize the root to be the attribute with the largest information gain
2. Initialize queue Q to contain the root
3. **While** Q is not empty **do** {
4.   Dequeue the first node $\hat{A}$ from Q
5.   **For each** attribute $B[i]$ (for i = 1, ..., n), evaluate splits on attribute $B[i]$
6.   Find the best split among these $B[i]$'s
7.   Use the best split to split node $\hat{A}$ into $\hat{A}_1$, $\hat{A}_2, \ldots, \hat{A}_k$
8.   For $i = 1, \ldots, k$, add $\hat{A}_i$ to Q if $\hat{A}_i$ is not well classified
9. }

3.3.4 How to Find the Best Split For Attribute $\hat{A}$

For the simplicity purpose, we only describe the Entropy splitting scheme, but our method is general, and can be used for Gini Index and other similar splitting schemes.

To find the best split for each node, we need to find the largest information gain for each node. Let S represent the set of the data belonging to the current node $\hat{A}$. Let R represent the set of requirements that each record in the current node has to satisfy.

To evaluation the information gain for the attribute $B[i]$, if all the attributes involved in R and $B[i]$ belong to the same party, this party can find the information gain for $B[i]$ by itself. However, except for the root node, it is unlikely that R and $B[i]$ belong to the same party; therefore, neither party can compute the information gain by itself. In this case, we are facing two challenges: one is to compute Entropy(S), and the other is to compute Gain(S, $B[i]$) for each candidate attribute $B[i]$.

First, let us compute Entropy(S). We divide R into two parts, R_a and R_b, where R_a represents the subset of the requirements that only involve Alice's attributes, and R_b represents the subset of requirements that only involve Bob's attributes. For instance, in Figure 2, suppose that S contains all the records whose Outlook = *Rain* and Humidity = *High*, so R_a = {Outlook = *Rain*} and R_b = {Humidity = *High*}.

Let V_a be a vector of size N. $V_a(i) = 1$ if the ith record satisfies R_a; $V_a(i) = 0$ otherwise. Because R_a belongs to Alice, Alice can compute V_a from her own share of attributes. Similarly, let V_b be a vector of size N. $V_b(i) = 1$ if the ith data item satisfies R_b; $V_b(i) = 0$ otherwise. Bob can compute V_b from his own share of attributes. Let V_j be a vector of size N, $V_j(i) = 1$ if the ith data item belongs to class j; $V_j(i) = 0$ otherwise. Both Alice and Bob can compute V_j.

Notes that a nonzero entry of $V = V_a \wedge V_b$ (i.e. $V(i) = V_a(i) \wedge V_b(i)$ for $i = 1, \ldots, N$) means the corresponding record satisfies both R_a and R_b, thus belonging to partition S. To build decision trees, we need to find out how many entries in V are non-zero. This is equivalent to computing the scalar product of V_a and V_b:

$$V_a \cdot V_b = \sum_{i=1}^{N} V_a(i) * V_b(i) \qquad (5)$$

However, Alice should not disclose her private data V_a to Bob, neither should Bob. We have developed a Scalar Product protocol (Section 4) that enables Alice and Bob to compute $V_a \cdot V_b$ without sharing information between each other.

After getting V_a, V_b, and V_j, we can now compute $\hat{P}_j$, where $\hat{P}_j$ is the number of occurrences of class j in partition S.

$$\hat{P}_j = V_a \cdot (V_b \wedge V_j) = (V_a \wedge V_j) \cdot V_b \qquad (6)$$

After knowing $\hat{P}_j$, we can compute $|S| = \sum_{j=1}^{n} \hat{P}_j$ and P_j, the relative frequency of class j in S:

$$P_j = \frac{\hat{P}_j}{|S|} \tag{7}$$

Therefore, we can compute Entropy(S) using Equation (1). We repeat the above computation for partition S_v to obtain $|S_v|$ and Entropy(S_v) for all values v of attribute $B[i]$. We then compute Gain(S, $B[i]$) using Equation (3).

The above process will be repeated until we get the information gain for all attributes $B[i]$ for $i = 1, \ldots, n$. Finally, we choose attribute $B[k]$ for the partition attribute for the current node of the tree, where Gain(S, $B[k]$) = max{Gain(S, $B[1]$), $\ldots$, Gain(S, $B[n]$)}.

We use an example to illustrate how to calculate the information gain using entropy (using gini index is similar). Figure 2 depicts Alice's and Bob's raw data, and S represents the whole data set.

Alice can compute $Gain(S, Outlook)$:

$Entropy(S)$

$= -\frac{3}{5}\log(\frac{3}{5}) - \frac{2}{5}\log(\frac{2}{5})$

$= 0.971$

$Gain(S, Outlook) = Entropy(S)$

$-\frac{2}{5}Entropy(S, Sunny) - \frac{3}{5}Entropy(S, Rain)$

$= 0.42$

Bob can compute $Gain(S, Humidity)$ and $Gain(S, Wind)$:

$Entropy(S) = 0.971$

$Gain(S, Humidity) = Entropy(S)$

$-\frac{3}{5}Entropy(S, High) - \frac{2}{5}Entropy(S, Normal)$

$= 0.02$

$Gain(S, Wind) = Entropy(S)$
$-\frac{3}{5}Entropy(S, Weak) - \frac{2}{5}Entropy(S, Strong)$

$= 0.42$

Alice and Bob can exchange these values and select the biggest one as the root. Suppose they choose Outlook as the root node, we will show how to compute Gain(S_{Rain}, Humidity) in order to build the sub trees. Where S_{Rain} is the set of data whose values of Outlook attribute are *Rain*.

Alice computes vector $V_a(Rain)$ where she sets the value to 1 if the value of attribute outlook is *Rain*; she sets the value to 0 otherwise. She then computes vector $V_a(Rain\_No)$, where she sets the value to 1 if the value of attribute outlook is *Rain* and the class label is *No*; she sets the value to 0 otherwise. Similarly, she computes vector $V_a(Rain\_Yes)$, where she sets the value to 1 if the value of attribute outlook is *Rain* and the class label is *Yes*; she sets the value to 0 otherwise. Therefore, Alice obtains:

$V_a(Rain) \quad = \quad (0,0,1,1,1)^T$

$V_a(Rain\_No) \quad = \quad (0,0,0,0,1)^T$

$V_a(Rain\_Yes) \quad = \quad (0,0,1,1,0)^T$

Bob computes $V_b(High)$ vector, where he sets the value to 1 if the value of attribute Humidity is *High*; he sets the value to 0 otherwise. Similarly Bob computes $V_b(Normal)$ vector, where he sets the value to 1 if the value of attribute Humidity is *Normal*; he sets the value to 0 otherwise. Therefore, Bob obtains:

$V_b(High) \quad = \quad (1,1,1,0,0)^T$

$V_b(Normal) \quad = \quad (0,0,0,1,1)^T$

$Gain(S_{Rain}, Humidity) = Entropy(S_{Rain})$

$-\frac{|S_{v=High}|}{|S_{Rain}|}Entropy(S_{Rain}, High)$

$-\frac{|S_{v=Normal}|}{|S_{Rain}|}Entropy(S_{Rain}, Normal)$

Since Alice knows all the necessary information to compute Entropy(S_{Rain}), Alice can compute it by herself. In what follows, $\hat{P}_{No}$ is the number of occurrences of class *No* in partition S_{Rain}; $\hat{P}_{Yes}$ is the number of occurrences of class *Yes* in partition S_{Rain}.

$|S_{Rain}| = 3$

$Entropy(S_{Rain})$

$= -\frac{\hat{P}_{No}}{|S_{Rain}|}\log(\frac{\hat{P}_{No}}{|S_{Rain}|})$

$-\frac{\hat{P}_{Yes}}{|S_{Rain}|}\log(\frac{\hat{P}_{Yes}}{|S_{Rain}|})$

$= -\frac{1}{|3}\log(\frac{1}{3}) - \frac{2}{3}\log(\frac{2}{3})$

$= 0.918$

In order to compute Entropy(S_{Rain},High), Alice and Bob has to collaborate because Alice knows S_{Rain}, but knows nothing about whether the Humidity is High or not, while Bob knows the Humidity attribute, but knows nothing about which record belong to S_{Rain}. In what follows, $\hat{P}_{No}$ is the number of occurrences of class *No* in partition S_{Rain} when Humidity = High; $\hat{P}_{Yes}$ is the number of occurrences of class *Yes* in partition S_{Rain} when Humidity = High.

Alice		
Day	Outlook	Play Ball
D1	Sunny	No
D2	Sunny	No
D3	Rain	Yes
D4	Rain	Yes
D5	Rain	No

Bob		
Day	Humidity	Wind	Play Ball
D1	High	Weak	No
D2	High	Strong	No
D3	High	Weak	Yes
D4	Normal	Weak	Yes
D5	Normal	Strong	No

Figure 2: The Original Data Set

$$|S_{v=High}| = V_a(Rain) \cdot V_b(High) = 1$$

$$\hat{P}_{No} = V_a(Rain\_No) \cdot V_b(High) = 0$$

$$\hat{P}_{Yes} = V_a(Rain\_Yes) \cdot V_b(High) = 1$$

$$Entropy(S_{Rain}, High)$$

$$= -\frac{\hat{P}_{No}}{|S_{v=High}|} \log(\frac{\hat{P}_{No}}{|S_{v=High}|})$$

$$-\frac{\hat{P}_{Yes}}{|S_{v=High}|} \log(\frac{\hat{P}_{Yes}}{|S_{v=High}|})$$

$$= 0$$

Similarly, Alice and Bob can compute Entropy(S_{Rain}, Normal). In what follows, $\hat{P}_{No}$ is the number of occurrences of class *No* in partition S_{Rain} when Humidity = Normal; $\hat{P}_{Yes}$ is the number of occurrences of class *Yes* in partition S_{Rain} when Humidity = Normal.

$$|S_{v=Normal}| = V_a(Rain) \cdot V_b(Normal) = 2$$

$$\hat{P}_{No} = V_a(Rain\_No) \cdot V_b(Normal) = 1$$

$$\hat{P}_{Yes} = V_a(Rain\_Yes) \cdot V_b(Normal) = 1$$

$$Entropy(S_{Rain}, Normal)$$

$$= -\frac{\hat{P}_{No}}{|S_{v=Normal}|} \log(\frac{\hat{P}_{No}}{|S_{v=Normal}|})$$

$$-\frac{\hat{P}_{Yes}}{|S_{v=Normal}|} \log(\frac{\hat{P}_{Yes}}{|S_{v=Normal}|})$$

$$= 1$$

Therefore, they can get

$$Gain(S_{Rain}, Humidity) = 0.252$$

Notice that the above computation is based on a series of secure scalar product protocols, which guarantees that Alice's private data ($V_a(Rain)$, $V_a(Rain\_No)$, and $V_a(Rain\_Yes)$) are not disclosed to Bob, and Bob's private data ($V_b(High)$ and $V_b(Normal)$) are not disclosed to Alice. In the same way, we can compute Gain(S_{Rain}, Wind). Then we can choose the attribute that causes the largest information gain, and use this attribute to partition the current node. This process can be repeated until we finish building the tree.

3.4 Tree Pruning

To use any existing pruning scheme, we need to know how to classify a sample data item using the decision tree. Once we can classify a sample data item, we can measure the accuracy of the decision tree, and hence being able to prune the tree using the schemes proposed in the literature (Rastogi & Shim 2000). Therefore, we need to solve the following problem:

Problem 3.2 Given a data item $d = (x_1, \ldots, x_s, y_1, \ldots, y_t)$, with $(x_1, \ldots, x_s)$ known by Alice and $(y_1, \ldots, y_t)$ known by Bob. How to use the decision tree T (known to both) to classify this data item.

Let us number the leaf nodes from 1 to L. We define three vectors: T_a, T_b, and C, all of which are of size L. Entries of T_a and T_b are initiated to 0. We also number the class labels from 1 to m, and initiate the entry i (for $i = 1, \ldots, L$) of C to k if the ith leaf node represents a category k.

Alice traverses the tree T. If a node is split using an Alice's attribute, Alice traverses to the corresponding child based on d. If a node is split using a Bob's attribute, Alice traverses to all children of the node. If Alice reaches a leaf node i, she changes the ith entry of T_a to 1. At the end, vector T_a records all the leaf nodes that Alice can reach based on her share of the data item d.

Bob does the similar: If a node is split using a Bob's attribute, he traverses to the corresponding child; otherwise, he traverses to all of children of the node. At the end, vector T_b records all the leaf nodes that Bob can reach based his share of the data item d.

It should be noted that $T = T_a \wedge T_b$ has one and only one non-zero entry because there is one and only one leaf node that both Alice and Bob can reach. Therefore, $T_a \wedge (T_b \wedge C)$ should also have one and only one non-zero entry, and the value of this entry equals to the index of the category that d is classified into. If we conduct the scalar product between T_a and $(T_b \wedge C)$, we should get the index of the category. Our Scalar Product protocol can allow Alice and Bob to conduct the scalar product between T_a and $(T_b \wedge C)$ without each party disclosing the private inputs to the other party.

4 Building Blocks

The security of our classification method is based on how two parties could compute the scalar product of their private vectors. The scalar product protocol described in the next sub-section is the main technical tool used in this paper. We will describe an efficient

solution of the scalar product problem based on the commodity server, an semi-trusted third party.

4.1 Introducing Commodity Server

For performance reasons, we use the help from an extra server, the commodity server, belonging to a third party. Alice and Bob could send request to the commodity server and receive data (called *commodities*) from the server, but the commodities should be independent of Alice's or Bob's private data. The purpose of the commodities is to help Alice and Bob conduct the desired computation.

The third party is semi-trusted in the following senses: (1) The third party should not be trusted; therefore it should not be possible to derive the private information of the data from Alice or Bob; it should not learn the computation result either. (2) The third party should not collude with both Alice and Bob. (3) The third party follows the protocol correctly. Because of these attributes, we say that the third party is an semi-trusted party. In real world, finding such a semi-trusted third party is much easier than finding a trusted third party.

As we will see from our solutions, the commodity server does not participate in the actual computation between Alice and Bob, it only supplies commodities that are independent of Alice and Bob's private data. Therefore, the server can even generate independent data off-line beforehand, and sell them as commodities to the prover and the verifier (hence the name "commodity server").

The commodity server model was first proposed by Beaver (Beaver 1997, Beaver 1998), and has been used for solving Private Information Retrieval problems in the literature (Beaver 1997, Beaver 1998, Di-Crescenzo, Ishai & Ostrovsky 1998) and various privacy preserving computation problems (Du 2001, Du & Zhan 2002).

4.2 Scalar Product

Alice has a vector A and Bob has another vector B, both of the vectors have n elements. Alice and Bob want to compute the scalar product between A and B, such that Alice gets V_1 and Bob gets V_2, where $V_1 + V_2 = A \cdot B$ and V_2 is randomly generated by Bob. Namely, the scalar product of A and B is divided into two secret pieces, with one piece going to Alice and the other going to Bob. We assume that the following computation is based on the real domain.

Protocol 4.1 *(Scalar Product Protocol)*

1. The Commodity Server generates two random vectors R_a and R_b of size n, and lets $r_a + r_b = R_a \cdot R_b$, where r_a (or r_b) is a randomly generated number. Then the server sends (R_a, r_a) to Alice, and (R_b, r_b) to Bob.

2. Alice sends $\hat{A} = A + R_a$ to Bob, and Bob sends $\hat{B} = B + R_b$ to Alice.

3. Bob generates a random number V_2, and computes $\hat{A} \cdot B + (r_b - V_2)$, then sends the result to Alice.

4. Alice computes $(\hat{A} \cdot B + (r_b - V_2)) - (R_a \cdot \hat{B}) + r_a = A \cdot B - V_2 + (r_b - R_a \cdot R_b + r_a) = A \cdot B - V_2 = V_1$.

The communication cost of this protocol is $4n$, which is 4 times more expensive than the *optimal* cost of a two-party scalar product (the optimal cost of a scalar product is defined as the cost of conducting the

product of A and B without the privacy constraints, namely one party just sends its data in plain to the other party). The cost can be further improved to $2n$ because the vectors R_a and R_b are random generated by the commodity server; therefore the commodity server can send just the seeds (numbers of constant size) to Alice and Bob, and the seeds can be used to compute the random vector. Solutions to the scalar product protocol have been proposed before by Atallah and Du (Atallah & Du 2001) and by Vaidya and Clifton (Vaidya & Clifton 2002). Both of these solutions achieves the security without using a third party; However, the communication of Atallah and Du's solution is much more expensive than this solution. Although Vaidya and Clifton's solution has similar communication cost as ours, its computation cost $O(n^2)$ is more expensive than ours $O(n)$, where n is the size of the vectors.

Theorem 4.1 *Protocol 4.1 does not allow Alice to learn B, it does not allow Bob to learn A either.*

Proof 4.1 *Since $\hat{A} = A + R_a$ is all what Bob gets, because of the randomness and the secrecy of R_a, Bob cannot find out A.*

Let us first assume that all the random numbers are generated from the real domain. According to the protocol, Alice gets (1) $\hat{B} = B + R_b$, (2) $Z = \hat{A} \cdot B + (r_b - V_2)$, and (3) r_a, R_a, where $r_a + r_b = R_a \cdot R_b$. We will show that for any arbitrary B', there exists r'_b, R'_b and V'_2 that satisfies the above equations.

Assume B' is an arbitrary vector. Let $R'_b = \hat{B} - B'$, $r'_b = R_a \cdot R_b - r_a$, and $V'_2 = \hat{A} \cdot B' + r'_b$. Therefore Alice has (1) $\hat{B} = B' + R'_b$, (2) $Z = \hat{A} \cdot B' + (r'_b - V'_2)$, and (3) r_a, R_a, where $r_a + r'_b = R_a \cdot R'_b$. Therefore from what Alice learns, there exists infinite possible values for B.

If the random numbers are not generated from the real domain, Alice might get some information about B. For example, if the elements of B are in the domain of $[0, 100]$, and we also know the random numbers are generated from $[0, 200]$ domain, then if an element of vector $B + R_b$ is 250, we know the original element in vector B is bigger than 50.

It should also be noted that our protocol does not deal with the situation where one party lies about its input. For example, instead sending $B + R_b$, Bob sends $B' + R_b$, where B' is an arbitrary vector. In that case, neither of them can get correct results, but as we have shown, neither of them can gain information about the other party's private input either.

In the last section, we did the computation based on the assumption that Bob let V_2=0. By doing so, the performance becomes better. However, it allows a malicious party to gain partial information. For example, if Bob is malicious and wants to know whether the first entry of V_a is 1, he can make up a V_b, such that only the first entry is 1, and the rests are 0's. If the scalar product result of V_a and V_b equals to 1, Bob knows the first entry of V_a is 1, otherwise it is 0.

If Bob does not let V_2=0, The scalar product (Protocol 4.1) can allow Alice to learn a_j and Bob to learn b_j, where $a_j + b_j = P_j$. Therefore nobody can know the actual value of P_j. The next section will show how to compute the gini index and the entropy without disclosing P_j to either Alice or Bob.
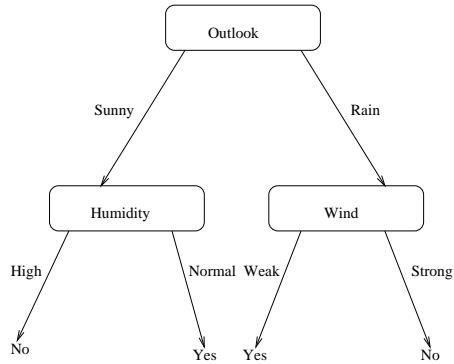
Figure 3: An Example Decision Tree

5 Privacy Improvement

5.1 Computing Gini Index

Problem 5.1 *(Gini Index)* Alice has a sequence $a_1, \ldots, a_m$, Bob has a sequence $b_1, \ldots, b_m$, and $P_j = a_j + b_j$ for $j = 1, \ldots, m$. Without disclosing each party's private inputs to the other, they want to compute

$$Gini(S) = 1 - \sum_{j=1}^{m}(a_j + b_j)^2.$$

Because of the following equations, the gini index can be actually computed using the Scalar Product protocol.

$$\sum_{j=1}^{m}(a_j + b_j)^2 = \sum_{j=1}^{m}a_j^2 + \sum_{j=1}^{m}b_j^2 + (a_1, \ldots, a_m) \cdot (b_1, \ldots, b_m)$$

5.2 Computing Entropy

Problem 5.2 *(Entropy)* Alice has a sequence $a_1, \ldots, a_m$, Bob has a sequence $b_1, \ldots, b_m$, and $P_j = a_j + b_j$ for $j = 1, \ldots, m$. Without disclosing each party's private inputs to the other, they want to compute

$$Entropy(S) = -\sum_{j=1}^{m}(a_j + b_j)\log(a_j + b_j)$$

To compute the entropy, the difficulty is how to compute $\log(a_j + b_j)$. First, let us look at the following *Logarithm* problem:

Problem 5.3 *(Logarithm)* Alice has a number a, and Bob has a number b. They want to compute $\log(a + b)$, such that Alice (only Alice) gets a', and Bob (only Bob) gets b', where b' is randomly generated and $a' + b' = \log(a + b)$. Nobody should be able to learn the other party's private input.

To solve this problem, Bob can generate a random positive number r in a finite real domain, and let $b' = -\log r$. Now Alice and Bob use the scalar product protocol to compute $r(a + b)$, such that only Alice knows $r(a + b)$. Because of r, Alice does not know b. Then Alice can compute $\log r(a+b) = \log r + \log(a + b) = a'$; therefore $a' + b' = \log(a + b)$.

Let $a'_j + b'_j = \log(a_j + b_j)$, where a'_j and b'_j are generated using the above Logarithm protocol. Now Alice and Bob need to compute the following, which

can also be achieved by using the scalar product protocol:

$$\sum_{j=1}^{m}(a_j + b_j)(a'_j + b'_j)$$

6 Information Disclosure Analysis

Information disclosure of privacy preserving data mining could come from two sources: one is the disclosure caused by the algorithm, the other is the disclosure caused by the result. The second type is inherent to the privacy preserving data mining problem. We have analyzed the first part of information disclosure when we discuss about our solution. Here, we discuss the second type of information disclosure. For that we assume the underlying algorithm itself is perfectly secure and discloses no information.

Assume the following decision tree model (Figure 3) is what we get after the decision tree building. We assume this model is 100% accurate. Now given a data item T_1 (Alice:(Sunny, No), Bob:(High, Weak, No)), we will show how Bob can figure out Alice's private data.

From the model, the only path that matches Bob's part of the data is the leftmost path. It is No→ High → Humidity→Sunny → Outlook. Therefore, Bob knows that Alice's value of outlook attribute is *Sunny*.

In practice, decision tree for the classification is much more complex, the tree is much deeper, and the accuracy of the model is not perfect. It is not as easy to guess the other party's private data as in this case. In the next step of our research, we plan to investigate how much information could be disclosed in practice from the final decision tree.

7 Conclusion and Future Work

Classification is an important problem in data mining. Although classification has been studied extensively in the past, the various techniques proposed for classification do not work for situations where the data are vertically partitioned: one piece is known by one party, the other piece by another party, and neither party wants to disclose their private pieces to the other party. We presented a solution to this problem using a semi-trusted commodity server. We discussed the security of our solution and the possible security problem inherent in the decision tree classification method.

One thing that we plan to investigate is how much information about the vertically partitioned data could be disclosed if both parties know the final

decision tree. This is very important to understand the limit of the privacy preserving classification. We will also investigate other decision tree building algorithms, and see whether our techniques can be applied to them or not.

References

Agrawal, D. & Aggarwal, C. (2001), On the design and quantification of privacy preserving data mining algorithms, *in* 'Proccedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems', Santa Barbara, California, USA.

Agrawal, R. & Srikant, R. (2000), Privacy-preserving data mining, *in* 'Proceedings of the 2000 ACM SIGMOD on Management of Data', Dallas, TX USA, pp. 439–450.

Atallah, M. J. & Du, W. (2001), Secure multi-party computational geometry, *in* 'WADS2001: 7th International Workshop on Algorithms and Data Structures', Providence, Rhode Island, USA, pp. 165–179.

Beaver, D. (1997), Commodity-based cryptography (extended abstract), *in* 'Proceedings of the 29th Annual ACM Symposium on Theory of Computing', El Paso, TX USA.

Beaver, D. (1998), Server-assisted cryptography, *in* 'Proceedings of the 1998 New Security Paradigms Workshop', Charlottesville, VA USA.

Di-Crescenzo, G., Ishai, Y. & Ostrovsky, R. (1998), Universal service-providers for database private information retrieval, *in* 'Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing'.

Du, W. (2001), A Study of Several Specific Secure Two-party Computation Problems, PhD thesis, Purdue University, West Lafayette, Indiana.

Du, W. & Zhan, Z. (2002), A practical approach to solve secure multi-party computation problems, *in* 'Proceedings of New Security Paradigms Workshop (to appear)', Virginia Beach, virginia, USA.

Franklin, M., Galil, Z. & Yung, M. (1992), An overview of secure distributed computing, Technical Report TR CUCS-00892, Department of Computer Science, Columbia University.

Goldreich, O. (1998), 'Secure multi-party computation (working draft)', Available from http://www.wisdom.weizmann.ac.il/home/oded/ public_html/foc.html.

Goldreich, O., Micali, S. & Wigderson, A. (1987), How to play any mental game, *in* 'Proceedings of the 19th Annual ACM Symposium on Theory of Computing', pp. 218–229.

Goldwasser, S. (1997), Multi-party computations: Past and present, *in* 'Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing', Santa Barbara, CA USA.

Lindell, Y. & Pinkas, B. (2000), Privacy preserving data mining, *in* 'Advances in Cryptology - Crypto2000, Lecture Notes in Computer Science', Vol. 1880.

Rastogi, R. & Shim, K. (2000), '(public): A decision tree classifier that integrates building and pruning', *Data Mining and Knowledge Discovery* **4**(4), 315–344.

Vaidya, J. & Clifton, C. (2002), Privacy preserving association rule mining in vertically partitioned data, *in* 'Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining'.

Yao, A. C. (1982), Protocols for secure computations, *in* 'Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science'.