

# PRIVACY IN MULTI-AGENT LEARNING: SECURELY INDUCING A MULTI-AGENT DECISION TREE

Karl Tuyls <sup>a</sup>      Bart Kuijpers <sup>b</sup>

<sup>a</sup> *Institute for Knowledge and Agent Technology, Universiteit Maastricht, The Netherlands, k.tuyls@cs.unimaas.nl*

<sup>b</sup> *Theoretical Computer Science Group, Universiteit Hasselt, Belgium, bart.kuijpers@uhasselt.be*

## Abstract

*In this paper we study the problem of how multiple distributed agents can jointly induce a decision tree, such that each of them preserves privacy over its own data and item set and each of them holds a part of the learned tree. Our work is original in the following ways: First of all, we consider agents to maintain data sites and jointly induce a decision tree. This merely reflects reality, as this is unlikely to be done by humans. Secondly, we formalize what is meant by data which is horizontally or vertically distributed in the literature. Thirdly, we show significance of this problem to real world applications by means of a motivating example. Fourthly, we generalize the state of the art to a situation in which we consider data which is as well vertically as horizontally distributed, which we call grid distributed data. We discuss two different evaluation methods for preserving privacy ID3, namely, first merging data horizontally and developing vertically or first merging data vertically and next developing horizontally. Finally, the main contribution of this paper is that we show, by means of a complexity analysis, that the former evaluation method is the more efficient.*

## 1 Introduction

In recent years the application possibilities of Multi-Agent Systems (MAS) in combination with the internet, has attracted and inspired many scientists from different research areas such as computer science, bioinformatics, artificial intelligence and economics, to actively participate in this relatively young field. Over the last few years this has naturally lead to a growing interest in security or privacy issues in MAS. More precisely, it became clear that discovering knowledge through a combination of different distributed databases (for instance over the internet), raises important security issues.

In this paper we concentrate ourselves along the line of Multi-Agent Learning. More precisely, we try to answer the question how multiple distributed agents can jointly learn a distributed decision tree over their individual data sites, while maintaining privacy over their individual data. By privacy of an agent we mean its individual data tuples and the kind of information it collects. The straightforward manner to learn a decision tree over multiple data sites would be to collect them in a centralized warehouse. Although learning results usually do not violate privacy of individuals, it cannot be assured that an unauthorized agent will not access the centralized warehouse with some malevolent intentions to misuse gathered information for his own purposes during the learning process. Neither can it be guaranteed that, when data is partitioned over different sites and data is not encrypted, it is impossible to derive new knowledge about the other sites.

Highly related to the work in this paper is the field of data mining, in which an entire subfield is dedicated to privacy preserving datamining. However, usually they do not consider agents for their mining process and they do not consider data to be completely distributed. More precisely, over the past few years state of the art research in privacy preserving data mining has concentrated itself along two major lines: data which is *horizontally distributed* and data which is *vertically distributed*. Horizontally partitioned data is data which is homogeneously distributed, meaning that all data tuples yield over the same item or feature set. Essentially this boils down to different data sites collecting the same kind of information over different individuals. Consider for instance a supermarket chain which gathers information on the buying behavior of its customers. Typically, such a company has different branches, implying data to be

horizontally distributed. Vertically distributed data is data which is heterogeneously distributed. Basically this means that data is collected by different sites or parties on the same individuals but with differing item or feature sets. Consider for instance financial institutions as banks and credit card companies, they both collect data on customers having a credit card but with differing item sets.

In this paper, we consider data which is *both* horizontally and vertically distributed, which we will call *grid partitioned data*. To our knowledge, there has been no research up till now in computer science or related fields that considers grid distributed data. Learning a distributed decision tree over such data sites, which are guarded by agents with their own degree of autonomy, is what we call *Multi-Agent Decision Tree Learning*. This kind of situation seems highly relevant and significant to real world applications as will become clear from our motivating example in Section 2.

In our work, we propose a new algorithm to preserve privacy when constructing a decision tree for classification over grid partitioned data using ID3, involving multiple agents. Most closely related to this work is that of Lindell and Pinkas [6] who introduced a secure multi-party computation technique for classification using the ID3 algorithm over horizontally partitioned data and that of Du and Zhan [2] who introduced a protocol for making ID3 secure over vertically partitioned data. An important contribution of our work is to consider horizontally and vertically distributed data at the same time. Furthermore, we also believe this is highly significant as most real life vital multi-agent situations, consist of grid partitioned data. For grid partitioned data, we discuss two different evaluation methods for preserving privacy ID3, namely, first merging horizontally and developing vertically or first merging vertically and next developing horizontally. We show in Section 5 by means of a complexity analysis that the former is the most efficient. In the context of secure multiparty computation, the “semi-honest” and the “malicious” model [12, 3] are considered. In the former, all parties follow the protocol strictly, but are allowed to remember everything they encounter while executing the protocol and to use this information to compute information about the other parties, whereas in the malicious model the parties are allowed to cheat. We assume the semi-honest model in this paper.

The rest of this paper is structured as follows. In the next Section we provide a motivating example for grid partitioned data, illustrating the importance of efficiently dealing with grid partitioned data in multi-agent applications. In Section 3, we sketch the preliminaries as the ID3 algorithm and definitions of horizontally, vertically and grid distributed data. Section 4 introduces our algorithm. We discuss the two different evaluation methods for preserving privacy ID3 in Section 5. Section 6 concludes the paper.

## 2 Example on grid partitioned data

Typically for financial institutions as banks is that they offer their clients different services as a savings account, choice of credit card, Maestro and all kinds of investment possibilities as mortgages, stock investments, fund orders and so on. Of course a bank is interested in knowing which are good customers, which are bad ones and which are possible defrauders. Reasons are obvious, making profit and avoiding losses because of clients which are not credit worthy and show unreliable behavior. Gathering all kinds of financial data about their customers and their transactions can help them in identifying risky clients and possible defrauders, preventing huge financial losses. More precisely, by using good learning techniques it becomes possible to generalize over these gathered data sets and identify possible risks for future cases or transactions. Data is gathered by agents which can communicate with other agents maintaining other datasets. Typical for different branches is to gather the same kind (i.e. item sets) of data on different clients, implying that data is *horizontally* partitioned. A possible item set re-occurring at banks is illustrated in Table 1.

Cust. nr.	mortgage	account	salary	...	–saldo	...
A11	25.00	104.2	2.2	..	<i>no</i>	...
B12	3.0	1.001	3.2	..	<i>yes</i>	...
⋮				⋮		⋮

Table 1: A possible item set.

However by combining their data sets, it would become possible to derive knowledge, leading to a high level of precision in triggering fraudulent behavior, which they would not have reached for individually. Consider for instance a simplified rule  $X, Y \rightarrow F$ , meaning that if features  $X$  and  $Y$  are satisfied this implies a high chance the transaction is fraudulent. It is not imaginary that this association rule is known to bank  $A$  and unknown to bank  $B$ , simply because  $B$  has not enough (local) tuples to support this rule. There is a reasonable chance that by combining their databases, agents of sites  $A$  and  $B$  would have discovered association rules which globally hold and which they would have not discovered individually,

implying a greater accuracy in identifying defrauders. Although such a cooperative behavior could save them a great deal of money, none of them, as they are competitors, would be willing to share all its transactions and itemsets with one another for obvious reasons.

Although it is possible for banks to gather substantial data on their clients there is still room for more improvement. More precisely, a bank does not typically manage all the services it offers. For instance credit card transactions are managed by separate companies collaborating with banks. Despite this cooperation, neither of them will be too happy to exchange data and item or feature sets on their customers. Still if they would be willing to collaborate, this could lead to a higher precision in identifying fraudulent cases and all parties would benefit. In other words, the group of people having a credit card is usually involved in an investment of all possible kinds: mortgages, stock market, order funds etc. This implies that the group of individuals on which the credit card companies gather data is more or less the same as the group on which banks gather data concerning investments. This boils down to data which is *vertically* partitioned.

Summarizing, this example shows that it is not imaginary at all that data appears to be as well horizontally as vertically distributed in real life situations, which we call *grid partitioned data*. Note that we can easily extend the above example to contain more parties. We could for instance add tax services, interested in tracking people cheating on their taxes. Most importantly, the example illustrates that we do not only need to consider privacy preserving techniques for horizontally or vertically distributed data, but that it is highly significant for real life applications to consider the combination of both.

### 3 Preliminaries

We start this section with a subsection summarizing the ID3 algorithm. Then we continue with a subsection formalizing horizontally, vertically and grid distributed data. We continue with preliminaries on multi-party computation.

#### 3.1 The ID3 algorithm

The ID3 algorithm (Inducing Decision Trees) was originally introduced by Quinlan in [8] and is described below in Algorithm 1. Here we briefly recall the steps involved in the algorithm. For a thorough discussion of the algorithm we refer the interested reader to [7].

The input of ID3 is a finite data set of tuples containing (discrete or nominal) values for a finite number of attributes, one of which is called the class attribute (also called target class). ID3 induces a decision tree from an example set in a top-down manner. More precisely, the algorithm starts at the root node, choosing each time the attribute which separates the data most efficiently according to their target class. Then the algorithm creates a branch for each value of this attribute and continues from there by repeating the above process until all attributes are used. To determine which attribute is best in classifying the given data set, a measure from information theory is used, namely *information gain*. Information gain is defined as the expected reduction in *entropy*. Entropy measures the homogeneity of a data set. More formally, the entropy of a data set of tuples  $S$  is defined as:

$$entropy(S) = \sum_{i=1}^c -p_i \log_2 p_i \tag{1}$$

where  $c$  is the total number of different values the target class can take on and  $p_i$  is the proportion of tuples of the data set having target value  $i$ . The information gain of an attribute  $A$  is then defined as:

$$gain(S, A) = entropy(S) - \sum_v \frac{|S_v|}{|S|} entropy(S_v) \tag{2}$$

with  $S_v$  the subset of  $S$  with tuples having value  $v$  for attribute  $A$ .

#### 3.2 Horizontally, vertically and grid partitioned data

In this section we provide a formal definition of horizontally, vertically and grid partitioned data. We will use the projection operation as defined in relational algebra in database theory.

Suppose we have:

1. A relation (or data set)  $S$  over the schema  $I, A_1, \dots, A_k, C$  consisting of a finite number of tuples. The attribute  $I$  is supposed to be a key (i.e., contain identifiers) and is not considered as an attribute

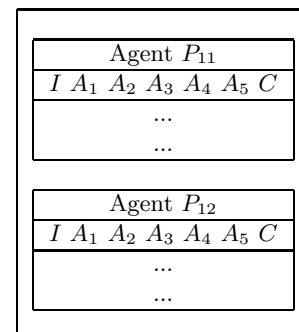


Figure 1: Horizontally distributed data.

---

**Algorithm 1** The ID3 Algorithm

---

**INPUT:**  $R$ , a set of attributes.**INPUT:**  $C$ , the class attribute.**INPUT:**  $S$ , data set of tuples.

- 1: **if**  $R$  is empty **then**
  - 2:   Return the leaf having the most frequent value in data set  $S$ .
  - 3: **else if** all tuples in  $S$  have the same class value **then**
  - 4:   Return a leaf with that specific class value.
  - 5: **else**
  - 6:   Determine attribute  $A$  with the highest information gain in  $S$ .
  - 7:   Partition  $S$  in  $m$  parts  $S(a_1), \dots, S(a_m)$  such that  $a_1, \dots, a_m$  are the different values of  $A$ .
  - 8:   Return a tree with root  $A$  and  $m$  branches labelled  $a_1 \dots a_m$ , such that branch  $i$  contains ID3( $R - \{A\}, C, S(a_i)$ ).
  - 9: **end if**
- 

to build the decision tree. The only purpose of the attribute  $I$  is to be able to join vertically distributed data. The attribute  $C$  will be referred to as the class attribute.

2. Agents  $P_{ij}$  with  $i = 1, \dots, r$ ,  $j = 1, \dots, s$  and  $r$  smaller than the number of attributes (i.e.,  $k + 1$ )

3. Each agent  $P_{ij}$  is holding a part  $S_{ij}$  containing information about certain attributes (including  $I$ ) and certain tuples. The  $S_{ij}$  are such that

- $S_{ij}$  is a partition of  $S$ , more precisely  $S = \cup_{j=1}^s \bowtie_{i=1}^r S_{ij}$ ;
- $S_{ij}$  and  $S_{ij'}$  have the same attributes but (parts of) different tuples of  $S$  when  $j \neq j'$ ;
- $S_{ij}$  and  $S_{i'j}$  have disjoint attributes but contain information about the same tuples of  $S$ .

**Definition** We call  $S$  *horizontally distributed* if and only if  $s = 1$ ; *vertically distributed* if and only if  $r = 1$ ; and *grid distributed* if and only if  $r, s \geq 2$ .

Agent $P_{11}$	Agent $P_{21}$
$I A_1 A_2 A_3$	$I A_4 A_5 C$
...	...
...	...

Figure 2: Vertically distributed data.

Examples of horizontally, vertically and grid distributed databases can be found in Figures 1, 2 and 3.

### 3.3 Preliminaries on multi-party computation

In this section we recall some results from multi-party computation that will be needed as building blocks in the algorithms in the next section.

Basically, secure multi-party computation (SMPC) makes sure that different agents involved in a computation process, do not learn anything more than the result(s) of the computation process and anything else that is derivable in a polynomial amount of time (without cheating). In the case of grid partitioned data in multi-agent decision tree learning, the learning process requires a lot of communication between the different agents. The SMPC techniques prevent any agent from deriving new knowledge about the other agents involved.

It is not our intention to give a complete overview here of SMPC, therefore we refer to [5, 12, 6, 10, 9, 1, 11]. Here we provide the security protocols necessary for our purposes, i.e., *the secure sum protocol, the Yao circuit, the secure union protocol, the secure size of set intersection protocol; and the  $x \ln(x)$  protocol.*

We remark that in the context of secure multi-party computation, two models, that we already mentioned before, are considered, namely the “semi-honest” and the “malicious” model [3, 12]. We will assume the semi-honest model in the description of our algorithms.

Agent $P_{11}$	Agent $P_{21}$	Agent $P_{31}$
$I A_1 A_2$	$I A_3$	$I A_4 A_5 C$
...	...	...
...	...	...
Agent $P_{12}$	Agent $P_{22}$	Agent $P_{32}$
$I A_1 A_2$	$I A_3$	$I A_4 A_5 C$
...	...	...
...	...	...
Agent $P_{13}$	Agent $P_{23}$	Agent $P_{33}$
$I A_1 A_2$	$I A_3$	$I A_4 A_5 C$
...	...	...
...	...	...

Figure 3: Grid distributed data.

### 3.3.1 Secure sum protocol

The goal of this protocol is that  $k > 2$  agents can compute the sum of the values each agent holds in such a way that no agent can learn anything about the values of the other agents.

The protocol of Kantarcioglu and Clifton [4] protects individual values by using a random number. Agent 0 adds a random number to its own value and sends it to Agent 1. Agent 1 cannot learn anything from this value due to the random number. Agent 1 adds his value to this number and sends it along to Agent 2. This process continues until the last agent has been reached. This agent adds his number to the number it received and sends it to Agent 0. Agent 0 can now compute the sum by distracting the random number from the sum it received of the last agent. Now Agent 0 reveals the sum to the other agents.

How safe is this protocol? It can be shown, by means of a polynomial time simulator, that, in the semi-honest model, this protocol is safe. Actually, to show safety it is necessary that all values remain within a finite domain  $[0, m]$  and all computations are done modulo  $m$ .

### 3.3.2 Yao circuit

Yao introduced in [12] the concept of *secure two agent computation*. He showed that any function  $f(x, y)$ , where  $x$  is the input of Agent 1 and  $y$  the input of Agent 2, can be evaluated in a secure way.

To formalize the concept of security, we concentrate on functions  $f$  (Yao makes use of Boolean circuits to represent a function  $f$ ) of the form  $f(x, y) = (f_1(x, y), f_2(x, y))$ . This function receives a part of its input, namely  $x$ , from Agent 1 and the other part of its input, namely  $y$ , from Agent 2. Agent 1 wants to learn  $f_1(x, y)$  and Agent 2 wants to learn  $f_2(x, y)$ . Suppose that protocol  $\Pi$  is used to learn  $f$ .  $View_i^\Pi$  is what Agent  $i$  learns by executing protocol  $\Pi$  and  $Output_i^\Pi$  is the output of Agent  $i$  ( $i = 1, 2$ ). Finally, let  $S_i$  be an algorithm that can be executed in polynomial time. Yao defines

$$\{S_1(x, f_1(x, y)), f_2(x, y)\} = \{View_1^\Pi(x, y), Output_2^\Pi(x, y)\}$$

and

$$\{f_1(x, y), S_2(y, f_2(x, y))\} = \{Output_1^\Pi(x, y), View_2^\Pi(x, y)\},$$

meaning that any agent can learn from  $f$ , by executing protocol  $\Pi$ , only those facts that can be learned in polynomial time from his/her input and his/her output. Executing the protocol does therefore not provide any extra information.

We remark that Goldreich *et al.* [3] generalized the results of Yao to more than two agents. Goldreich *et al.* also gave the composition theorem that states that if a function  $g$  can be reduced safely to a function  $f$ , and if there is a protocol to safely compute  $f$ , then also  $g$  can be computed safely.

In this paper, we will refer to this type of circuits as *Yao circuits*, even if they concern more than two agents.

### 3.3.3 Secure union protocol

When there are only two agents, computing the union of two sets belonging to each of those agents, this can lead to security problems. Indeed, the knowledge about ones own set and about the union, gives (at least partial) knowledge about the other agents set. In this section, we outline a method to compute the union of  $k$  itemsets, belonging to  $k$  agents,  $k > 2$ . The goal is that all agents should learn the union, without learning about the itemsets of other agents. The algorithm is from Kantarcioglu and Clifton [4] and consists of four phases that we sketch below. These authors also show its security.

*Phase 1:* All agents generate a commutative, deterministic encryption key  $E_i$  and a decryption key  $D_i$ . Each itemset is augmented with fake or dummy items (this is done to prevent the determination of the cardinality of the itemset). At the end of Phase 1, each agent has an itemset of the same size (which is agreed upon at the start).

*Phase 2:* Each agent encrypts its items and communicates them to the next agent (the communication is cyclic as in the case of secure sum computation, i.e., Agent  $i$  sends information to Agent  $(i + 1) \bmod k$ ). Each agent encrypts what he receives and passes it to the next agent. This continues until each Agent  $i$  is in the possession of the completely encrypted items of Agent  $(i + 1) \bmod k$ . We remark that to continue one more step would be no longer secure.

*Phase 3:* The even-numbered agents send all items in their possession to Agent 0 and the odd-numbered agents do the same to Agent 1 (the last agent always has to send to Agent 1 to avoid that a agent gets its own fully encrypted itemset). Agents 0 and 1 take the union of what they received and remove the doubles. Agent 1 sends everything he has to Agent 0, who removes the doubles. At this point the union is in the possession of Agent 0, but fully encrypted.

*Phase 4:* The encrypted union is sent to all agents to be decrypted. Finally the fake items are removed and the result is announced to all agents.

### 3.3.4 Secure size of set intersection protocol

When there are only two agents, computing the size of the intersection of two sets belonging to each of the agents can lead to security problems. Indeed, the knowledge about ones own set and about the size of the intersection of two sets gives (at least partial) knowledge about the set of the other agent. So, we are interested to compute the size of set intersection of  $k$  itemsets, belonging to  $k$  agents,  $k > 2$ . The goal is that all agents should learn the size of set intersection, without learning about the itemsets of other agents.

Jaideep Vaidya [11] proposed a protocol for the secure computation of the size of set intersection. It is similar to the secure union protocol, and we will not repeat the details here but refer to [11].

### 3.3.5 $x \ln(x)$ protocol

The  $x \ln(x)$  protocol, due to Lindell and Pinkas [6], is different from the previous protocols. It uses Yao circuits, as mentioned earlier in this section. Because circuits are only suitable for two agents, also this protocol is only suitable for two agents. Assume we have two agents, called Alice and Bob. Alice has a value  $x_a$  and Bob has a value  $x_b$ . The goal of the  $x \ln(x)$  protocol is to give Alice and Bob both a share  $s_a$  and  $s_b$  respectively, such that  $s_a + s_b = (x_a + x_b) \ln(x_a + x_b)$

The  $x \ln(x)$  protocol makes use of two subprotocols. The first receives two values  $x_a$  and  $x_b$  as input and returns two random shares of  $\ln(x_a + x_b)$  as output (using a Taylor series). The second, called the multiplication protocol, receives two values  $u_a$  and  $u_b$  as input and returns two random shares of  $u_a \cdot u_b$  as output. Alice and Bob run the  $\ln(x)$  protocol and become shares  $u_a$  and  $u_b$ . Next, the multiplication protocol is executed twice. First with  $u_a$  and  $x_b$  as input. This gives Alice and Bob respectively shares  $v_a$  and  $v_b$ . the second time it is called with  $x_a$  and  $u_b$ , giving Alice and Bob respectively shares  $w_a$  and  $w_b$ . Alice now has  $x_a$ ,  $u_a$ ,  $v_a$  and  $w_a$ , with which she can compute  $s_a = x_a u_a + v_a + w_a$ . Bob can construct  $s_b = x_b u_b + v_b + w_b$  in a similar way. Since  $x_a u_a + x_b u_b + x_a u_b + x_b u_a = (x_a + x_b)(u_a + u_b) = (x_a + x_b) \ln(x_a + x_b)$ , Alice and Bob both have their share of  $(x_a + x_b) \ln(x_a + x_b)$ .

## 4 Privacy preserving ID3: Grid partitioned data

In the present section, we introduce our algorithms, preserving privacy over grid partitioned data. Basically, we will study the following dilemma: when data is grid partitioned we can first merge it horizontally and then further develop the process vertically, or the other way around. Obviously other ways of doing this are possible as well, but we consider only the two straightforward ones.

In this paper we consider privacy as protecting individual data tuples as well as protecting attributes and values of attributes. So each agent will reveal as little as possible about its data while still constructing an applicable distributed decision tree. The only thing that is known about the tree by all agents is its structure and which agent is responsible for each decision node. More precisely, which agent possesses the attribute used to make the decision, but not which attribute (and value). We assume that only a limited number of agents know the class attribute and no agent knows the entire set of attributes, which is obvious as we use grid partitioned data.

Once the tree is constructed instance classification proceeds as follows. The agent that wishes to classify a new unseen instance knows the root node of the tree: or the node resides at his site or he knows the root node-identification (nodeID). A root node identification contains a code indentifying the agent possessing that particular node. Basically, when classifying a new instance, control passes from agent to agent, depending on the decision nodes that are visited. Every agent knows the tuples attribute values for the nodes at its site but knows nothing about the other attribute values. The classification then happens as in Algorithm 2.

---

**Algorithm 2** The classification algorithm called *classify*( $t, nodeID$ ). A site wishes to classify a new instance  $t$ . Control starts at the root node (which every agent knows.)

---

- 1: **if** The nodeID is a leaf node **then**
  - 2:   its classification value (or distribution) is returned.
  - 3: **else if** The nodeID is an interior node **then**
  - 4:   node = local node with nodeID
  - 5:   value = value of attribute node.A (used as decision attribute) for the tuple  $t$  we are classifying
  - 6:   childID = node.value
  - 7:   return childID.classify( $t, childID$ )
  - 8: **end if**
-

## 4.1 Grid Partitioned data

We will introduce the grid partitioned privacy preserving algorithm by running through the different steps of ID3 informally. It is important to realize that no site knows the complete attribute set  $S$  and only a limited number of agents know the class attribute, more particularly as much as there are horizontal distributions.

### 4.1.1 Horizontal merge and vertical development

Recall the ID3 algorithm from Section 3.1. We will separately consider its three basic steps, i.e. *emptiness test* of the attribute set  $S$ , all transactions having the same class label, i.e. *class label test* and the *default case*. Here we consider the case that we first merge the data horizontally and continue vertically. A horizontal merge means that we eliminate the horizontal distribution, leaving only a vertical distribution.

**Emptiness test** To determine if there are any attributes left, as many agents as there are vertical distributions need to cooperate with one another as we need to know all attributes to compute this test. This can be easily understood from Figure 3. More precisely, in the example of that figure, agents  $agent_{11}$ ,  $agent_{21}$  and  $agent_{31}$  can determine together if there are any attributes left. These agents check how many possible attributes they still possess as a candidate decision node and pass this value as input to the secure sum protocol. At the end of the protocol the sum and the random value are passed to a Yao circuit, which tests if the sum equals zero (meaning that  $S$  is empty) or not.

In case of the sum being zero, the most frequent class value has to be determined. This is done in the following manner: first all agents determine the tuples reaching the current node in the tree. Then these tuples are merged horizontally by constructing a union over the vertical groups (over index  $i$ ), i.e. for each vertical group a secure union protocol is applied. In case of Figure 3 we have the following groups computing a union:  $agent_{11}$ ,  $agent_{12}$  and  $agent_{13}$ ;  $agent_{21}$ ,  $agent_{22}$  and  $agent_{23}$  and  $agent_{31}$ ,  $agent_{32}$  and  $agent_{33}$ .

Agents which are located on the same horizontal layer (meaning that they have the same index for  $j$ ), which are vertically distributed, will use the same encryption key to compute the vertical unions (i.e. the horizontal merge). In our example these are,  $agent_{11}$ ,  $agent_{21}$  and  $agent_{31}$ ;  $agent_{12}$ ,  $agent_{22}$  and  $agent_{32}$ ;  $agent_{13}$ ,  $agent_{23}$  and  $agent_{33}$ . At this stage there is only a vertical distribution left over the entire distributed database, as we *merged* data horizontally. Now we will continue by *developing vertically*. The intersection of these different sets with the tuples in a particular class give the number of tuples that reach that point in the tree. This can be done for each class value; note that it is not necessary to use the secure size of set protocol because the unions are already encrypted. This gives us eventually the most frequent class value. Note that it is not necessary to decrypt the values again to compute the intersections. The reason is that we used the same encryption keys for agents at the same horizontal level, implying that equal values in the encrypted unions are also equal in the real unions. Now a leaf can be constructed with a certain leaf identifier. The value of the leaf is known by the agents that have the class attribute. The others only know the identifier.

**Class label test** Checking whether all transactions in the training set  $S$  have the same class value happens analogously to determining the most frequent class value. More precisely, one agent knowing the class attribute, can compute the possible intersections, i.e., the intersections of the sets of tuples which might reach the current level of the tree or node of the tree with the tuples in a particular class. If all intersections equal zero besides one, all tuples in  $S$  have that particular class value. Since they are all the same, now a leaf node is constructed. The agents which were joined in one vertical group knowing the class attribute, all know the id of the node (*nodeID* in the algorithms) and the specific class value. All other agents just get to know the nodeID of the node, which they need in case they need to classify a new tuple leading to this node in the tree.

**Default case** In this case, the best classifying attribute has to be determined. To do this, transactions or tuples need to be counted. To learn these numbers recall that entropy and information gain were defined as follows:  $entropy(S) = \sum_{i=1}^c -p_i \log_2 p_i$  where  $c$  is the total number of different values the target class can take on and  $p_i$  is the proportion of tuples of the data set having target value  $i$ , i.e.  $\frac{N_i}{N}$ , where  $N$  is the total number of tuples reaching the current node and  $N_i$  is the number of tuples with attribute value  $a_i$ . The information gain of an attribute  $A$  is then defined as:  $gain(S, A) = entropy(S) - \sum_v \frac{|S_v|}{|S|} entropy(S_v)$

For each attribute information gain needs to be computed. First all agents determine the tuples reaching the current node in the tree, i.e.  $N$ . Then these tuples are merged horizontally by constructing a union over the vertical groups, i.e. for each vertical group a secure union protocol is applied. In case of Figure 3, we have the following groups computing a union:  $agent_{11}$ ,  $agent_{12}$  and  $agent_{13}$ ;  $agent_{21}$ ,  $agent_{22}$  and  $agent_{23}$ ;  $agent_{31}$ ,  $agent_{32}$  and  $agent_{33}$ . For every vertical group, one agent will iterate over its attributes.

For each such attribute, numbers of tuples need to be counted for every value of this attribute. Thus for every value of the attribute (which is encrypted), an intersection is computed over all the sets, resulting from the horizontal merge. When we know all these numbers, the information gain of this attribute can be computed. This step is repeated for all attributes. The process described so far is called *vertical development*. On of the agents of each vertical group saves the the information gain of its attributes. These agents can then cooperate to compute the best classifying one. Finally, the agent which possesses the best classifying attribute constructs a decision node with is given a node identifier *nodeID*. The value of the node is communicated to the other agents that also possess this attribute. The other agents only get to know the identifier.

The complete description can be found in Algorithm 3.

---

**Algorithm 3** The privacy preserving ID3 algorithm over grid partitioned data when data is merged horizontally and further developed vertically.

---

**INPUT:**  $R$ , The set of attributes distributed among the agents  $P_{ij}$  with  $i = 1, \dots, r, j = 1, \dots, s$ .

**INPUT:**  $C$ , The class attribute with  $d$  class values,  $c_1, \dots, c_d$ .

**INPUT:**  $S$ , the grid distributed data set over agents  $P_{ij}$  with  $i = 1, \dots, r, j = 1, \dots, s$  and agents  $P_{r,j}$  holding the class attribute .

- 1: **if** (Emptiness test)The agents test if  $R$  is empty **then**
- 2:   Secure sum protocol and Yao circuit are used to test whether  $R$  is empty.
- 3:   In case the attribute set is empty, Secure union protocol is used to merge data horizontally. For the vertical development, the secure size of set intersection protocol does NOT have to be used. A Yao circuit is used to calculate which class value  $c_i$  is most frequent. A leaf node with class value  $c_i$  is returned.
- 4: **else if** All agents test whether all tuples have the same class value  $c_i$  **then**
- 5:   Secure union protocol and Yao circuit are used to calculate this.
- 6:   In case the test is TRUE, a leaf with class value  $c_i$  is returned.
- 7: **else**
- 8:   Determine attribute  $A$ , classifying most accurately tuples in  $S$ : use secure union and secure sum protocols.
- 9:   Partition  $S$  in  $m$  parts  $S(a_1), \dots, S(a_m)$  such that  $a_1 \dots a_m$  are the different values of  $A$ .
- 10:   Return a tree with root  $A$  and  $m$  branches  $a_1 \dots a_m$  such that branch  $i$  contains  $ID3(R - \{A\}, C, S(a_i))$ .
- 11: **end if**

---

#### 4.1.2 Vertical merge and horizontal development

**Emptiness test** To determine if there are any attributes left, as many agents as there are vertical distributions need to cooperate with one another as we need to know all attributes to compute this test. So essentially this is done in the same manner as with the horizontal merge, i.e. the previous algorithm.

To determine the most frequent class value we will merge data vertically. More precisely, every agent first determines the number of tuples that reach the current level of the tree or node of the tree. For this the agents only use the attributes they possess. Then we merge vertically by letting cooperate the agents at the same horizontal level. In our example these are *agent*<sub>11</sub>, *agent*<sub>21</sub> and *agent*<sub>31</sub>; *agent*<sub>12</sub>, *agent*<sub>22</sub> and *agent*<sub>32</sub>; *agent*<sub>13</sub>, *agent*<sub>23</sub> and *agent*<sub>33</sub>. The agents that possess the class attribute now need to compute a set per class value. In our example these are agents *agent*<sub>31</sub>, *agent*<sub>32</sub> and *agent*<sub>33</sub>. They compute as many secure size of set protocols as there are class values. In this manner they compute per horizontal group (or vertical merge) the number of transactions per class value. If the agents possessing the class attributes have computed these intersections, they have to cooperate to find out the total number of tuples per class value. They compute this by using a secure sum protocol per class value. Then these values are passed on to a Yao circuit to be able to learn the most frequent class value. Now a leaf can be constructed with a certain leaf identifier. The value of the leaf is known by the agents that have the class attribute. The others only know the identifier.

**Class label test** Determining whether all tuples have the same class value is analogous to the previous step. The difference lies in the Yao circuit, which will test if all sums equal zero except for one. Again a leaf can be constructed with a certain leaf identifier. The value of the leaf is known by the agents that have the class attribute. The others only know the identifier.

**Default case** We need to compute the best classifying attribute. To do this, transactions or tuples need to be counted. To learn these numbers recall that entropy and information gain were defined as follows:  $entropy(S) = \sum_{i=1}^c -p_i \log_2 p_i$  where  $c$  is the total number of different values the target class can take on

and  $p_i$  is the proportion of tuples of the data set having target value  $i$ , i.e.  $\frac{N_i}{N}$ , where  $N$  is the total number of tuples reaching the current node and  $N_i$  is the number of tuples with attribute value  $a_i$ . The information gain of an attribute  $A$  is then defined as:  $gain(S, A) = entropy(S) - \sum_v \frac{|S_v|}{|S|} entropy(S_v)$

First, we merge data vertically. More precisely, every agent first determines the number of tuples that reach the current level or node of the tree. For this the agents only use the attributes they possess. Then data is merged vertically by agents at the same horizontal layer (having the same  $j$  index in  $P_{ij}$ ) via a secure size of set intersection protocol to obtain exactly those tuples that are in the current dataset associated to the node under consideration in the tree. In our example these are  $agent_{11}$ ,  $agent_{21}$  and  $agent_{31}$ ;  $agent_{12}$ ,  $agent_{22}$  and  $agent_{32}$ ;  $agent_{13}$ ,  $agent_{23}$  and  $agent_{33}$ . Through a secure sum protocol these numbers can now be added to learn the number of tuples that reach the current node of the tree, which is denoted by  $N$  in the entropy formula.

Now we need to compute for each remaining attribute its information gain. This is done by computing for each value of an attribute over each horizontal layer, the number of tuples having this attribute value (we compute  $N_i$ ). This is done by using the secure size of set protocol. Then these numbers can be added over all horizontal layers by using a secure sum protocol. This is called *horizontal development*. The secure sum (added with the random value) and the random value itself multiplied by one are provided to the  $xln(x)$  protocol. This circuit will then output shares of the result which then can be used as input to a circuit which securely computes its sum and outputs which attribute classifies the tuples best. A decision node can now be constructed for the best classifying attribute.

The description of the algorithm is summarized in Algorithm 4.

---

**Algorithm 4** The privacy preserving ID3 algorithm over grid partitioned data when data is merged vertically and further developed horizontally.

---

**INPUT:**  $R$ , The set of attributes distributed among the agents  $P_{ij}$  with  $i = 1, \dots, r, j = 1, \dots, s$ .

**INPUT:**  $C$ , The class attribute with  $d$  class values,  $c_1, \dots, c_d$ .

**INPUT:**  $S$ , the grid distributed data set over agents  $P_{ij}$  with  $i = 1, \dots, r, j = 1, \dots, s$  and agents  $P_{r,j}$  holding the class attribute .

- 1: **if** (Emptiness test)The agents test if  $R$  is empty **then**
  - 2:   Secure sum protocol and Yao circuit are used to test whether  $R$  is empty.
  - 3:   In case the attribute set is empty, Secure size of set intersection protocol, secure sum protocol and Yao circuit are used to calculate which class value  $c_i$  is most frequent. a leaf node with class value  $c_i$  is returned.
  - 4: **else if** All agents test whether all tuples have the same class value  $c_i$  **then**
  - 5:   Secure size of set intersection protocol, secure sum protocol and Yao circuit are used to calculate this.
  - 6:   In case the test is TRUE, a leaf with class value  $c_i$  is returned.
  - 7: **else**
  - 8:   Determine attribute  $A$ , classifying most accurately tuples in  $S$ : use secure size of intersection, secure sum and  $xln(x)$  protocols.
  - 9:   Partition  $S$  in  $m$  parts  $S(a_1), \dots, S(a_m)$  such that  $a_1 \dots a_m$  are the different values of  $A$ .
  - 10:   Return a tree with root  $A$  and  $m$  branches  $a_1 \dots a_m$  such that branch  $i$  contains  $ID3(R - \{A\}, C, S(a_i))$ .
  - 11: **end if**
- 

## 5 Complexity analysis

In this section we analyse the complexity of the two computation strategies proposed in the previous section: first merging horizontally and developing vertically or first merging vertically and next developing horizontally.

The different quantities  $k$ ,  $h$ ,  $v$ ,  $|T|$ ,  $|R|$ ,  $c$ ,  $i$ ,  $t$  and  $n$  that play a role in this analysis are explained in the next table.

The predominant task in the ID3 algorithm is to determine the attribute with the highest Information Gain and we will base our analysis mainly on this component.

Notation	Meaning
$k$	the number of agents
$h$	the number of horizontal groups
$v$	the number of vertical groups
$ T $	the number of tuples in the data set
$ R $	the number of attributes
$c$	the number of values for the class attribute $C$
$i$	the maximal number of values for an attribute
$t$	the maximal length of encryption keys
$n$	the maximal length of Taylor series

## 5.1 The complexity of the components from SMPC

In discussing the complexity of the building blocks described in Section 3.3 usually two components are considered: the *computational complexity* and the *communication complexity*. The former considers the cost of computations in the classical sense, the latter considers the cost of passing messages, e.g., between different agents.

### 5.1.1 The complexity of the secure sum protocol

For the secure sum protocol with  $k$  agents, the computation and communication costs are both  $O(k \log(|T|))$ . Each of the agents never outputs values larger than  $|T|$  and the messages passed are never larger than  $|T|$ . Assuming binary encoding of numbers, this gives the above result.

### 5.1.2 The complexity of the secure union protocol and secure size of intersection protocol

For the secure union protocol and the secure size of intersection protocol with  $k$  agents, the computation cost is  $O(k^2|T|t^3)$  and the communication cost is  $O(k^2|T|t)$ . Indeed, the agents send sets of at most size  $|T|$ , they make use of encryption keys of length  $t$ , hence the factor  $t^3$  in the computation cost, and every agent has to encrypt  $k^2$  sets. The value  $t$  in the communication cost points at the size of the sets that are transmitted. In total  $k^2$  messages are sent of size  $|T|t$ .

### 5.1.3 The complexity of the secure $x \ln x$ protocol and Yao circuits

The secure  $x \ln x$  protocol for two agents has a computational cost of  $O(\log(|T|))$  and a communication cost of  $O(n \log(|T|)t)$ . It takes input values of at most  $|T|$ . The protocol also depends on a value  $n$  that determines how far a Taylor series is developed. The protocol contains a Yao circuit, created by one of the agents who also gives his input to the circuit and passes it to the other agent. This explains the communication cost, in which  $n$  obviously plays a role since it determines the size of the circuit. The second agent receives the circuit and feeds his input to the circuit. Hereto one oblivious transfer is performed per bit. This step explains the computational cost.

## 5.2 The complexity of first horizontal merging

To determine the attribute that best classifies the data, for each attribute unions have to be determined over  $h$  agents. The exact number of unions may depend on the attribute under consideration. If it is an attribute that belongs to the agent that possesses the class attribute, it are  $v + c + i - 1$  unions. If an other agent belongs the class attribute, it are  $v + c + i - 2$  unions. Also the number of unions to be transmitted depends on the attribute. For attributes in the possession of the owner of the class attribute, there are  $v - 1$  unions to be transmitted, for other attributes  $v + c - 2$ .

So, we conclude:

$$\text{computation cost} = O(|R|(v + c + i)(h^2|T|t^3)) \text{ and}$$

$$\text{communication cost} = O(|R|(v + c)(h^2|T|t)).$$

We end this section with a remark on how this protocol could be made more efficient. Remark that the strength of this protocol resides in the fact that adjacent agents may use the same encryption key. For this reason it is not necessary to use the secure size of set intersection protocol to calculate intersections. The first phase of this protocol can be skipped because the same keys are used when computing unions. This is possible here because the data is both horizontally and vertically distributed. When the data is only vertically distributed, it would also be possible to let the agents agree on some encryption keys and to simplify the protocol in this way.

## 5.3 The complexity of first vertical merging

Per attribute  $1 + c + i + ci$  values have to be computed, namely: *The number of transactions reaching the current node: 1;* *The number of transactions reaching the current node per class value: c;* *The number of transactions reaching the current node per attribute value: i* and *The number of transactions reaching the current node per class value and per attribute value: c.i.*

All these values can be computed via a secure size of set intersection protocol that is each time executed by  $v$  agents. Since we have to count this for each horizontal group, this gives in total  $h(1 + c + i + ci)$  calls to the secure size of set intersection protocol. With these values the computation continues. In case

of two horizontal groups this is with the  $x \ln x$  protocol; in the case of more horizontal groups this is with the secure sum protocol, followed by the  $x \ln x$  protocol.

So, we conclude:

$$\text{computation cost} = O(|R|(h(1+c+i+ci))(v^2|T|t^3) + |R|(1+c+i+ci)(\log(|T|)) + |R| \log(|T|)) \quad [+O(h \log(|T|))] \text{ and}$$

$$\text{communication cost} = O(|R|(h(1+c+i+ci)).(v^2|T|t) + |R|(1+c+i+ci)n(\log(|T|)t) + |R| \log(|T|)t) \quad [+O(h \log(|T|))].$$

## 5.4 Conclusion on the complexity analysis

We start by remarking that it looks more logical to merge the data first horizontally and then to further develop it vertically. The emptiness test can be implemented more efficiently in the former case. The secure  $x \ln x$  protocol gives an approximated result, but the difference from the real result is small. This protocol also makes heavy use of circuit computations. In practice it is preferable to avoid this.

For what concerns complexity, the above obtained expressions also show that first horizontally merging is advantageous. And as remarked before it can be improved by an optimal use of encryption. Indeed, by giving different agents the same encryption key it is not necessary to perform the secure size of set intersection protocols after the secure union protocols have been executed.

## 6 Conclusions

In this paper we first discussed the significance of extending the current state of the art in distributed decision tree learning to grid partitioned data, i.e. data which is as well horizontally as vertically partitioned. Our motivating example shows that this situation is of great interest to real world situations and applications. Then we continued by formally defining horizontally, vertically and grid partitioned data. To our knowledge we are the first to formalize the concept of grid partitioned data. The main contributions of this paper are the two algorithms to securely induce a distributed decision tree when data is grid partitioned. More precisely, we considered two possible solutions: one in which data is first merged horizontally and then further developed vertically and vice versa. The complexity analysis of both algorithms shows that it is more efficient to first merge data horizontally and further develop it vertically than the other way around.

## References

- [1] C. Clifton and D. Marks. Security and privacy implications of data mining. In *Proceedings of the ACM SIGMOD Workshop on Data Mining and Knowledge Discovery*, 1996.
- [2] W. Du and Z. Zhan. Building decision tree classifier on private data. In *IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining*, pages 1–8, 2002.
- [3] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing*, pages 218–229, 1987.
- [4] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2002.
- [5] M. Kantarcioglu, C. Clifton, X. Lin, and M. Zhu. Tools for privacy-preserving distributed data mining. *SIGKDD Explorations*, 2003.
- [6] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Advances in Cryptology CRYPTO 2000*, 2000.
- [7] T. Mitchell. Machine learning. *McGraw-Hill Series in Computer Science*, 1997.
- [8] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
- [9] R. A. Ramakrishnan and Srikant. Privacy-preserving data mining. In *Proceedings of SIGMOD 2000*, 2000.
- [10] C. Su and K. Sakurai. Secure computation over distributed databases. *IPSJ journal*.
- [11] J. Vaidya. *Privacy Preserving Data Mining over Vertically Partitioned Data*. PhD thesis. Purdue University, 2004.
- [12] A. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*.