Privacy-Preserving Decision Trees over Vertically Partitioned Data*

Jaideep Vaidya[1] and Chris Clifton[2]

[1] MSIS Department, Rutgers University, Newark, NJ 07102
jsvaidya@rbs.rutgers.edu
http://cimic.rutgers.edu/~jsvaidya
[2] Department of Computer Science, Purdue University,
West Lafayette, IN 47907
clifton@cs.purdue.edu
http://www.cs.purdue.edu/people/clifton

Abstract. Privacy and security concerns can prevent sharing of data, derailing data mining projects. Distributed knowledge discovery, if done correctly, can alleviate this problem. In this paper, we tackle the problem of classification. We introduce a generalized privacy preserving variant of the ID3 algorithm for vertically partitioned data distributed over two or more parties. Along with the algorithm, we give a complete proof of security that gives a tight bound on the information revealed.

1 Introduction

There has been growing interest in privacy-preserving data mining since the seminal papers in 2000 [1,2]. Classification is one of the most ubiquitous data mining problems found in real life. Decision tree classification is one of the best known solution approaches. ID3, first proposed by Quinlan[3] is a particularly elegant and intuitive solution. This paper presents an algorithm for privately building an ID3 decision tree. While this has been done for horizontally partitioned data [4], we present an algorithm for *vertically partitioned* data: a portion of each instance is present at each site, but no site contains complete information for any instance. This problem has been addressed[5], but the solution is limited to the case where both parties have the class attribute. In addition, both the previous methods are limited to two parties. The method presented here works for any number of parties, and the class attribute (or other attributes) need be known only to one party. Our method is trivially extendible to the simplified case where all parties know the class attribute.

There has been other work in privacy-preserving data mining. One approach is to add "noise" to the data before the data mining process, and using techniques that mitigate the impact of the noise from the data mining results[1,6,7,8]. However, recently there has been debate about the security properties of such algorithms [9].

* This material is based upon work supported by the National Science Foundation under Grant No. 0312357.

S. Jajodia and D. Wijesekera (Eds.): Data and Applications Security 2005, LNCS 3654, pp. 139–152, 2005.

Other work follows the secure multiparty computation approach found in cryptography, achieving "perfect" privacy, i.e., nothing is learned that could not be deduced from one's own data and the results. This includes Lindell's work [2], as well as work on association rule mining [10,11,12,13], clustering [14,15], and some work on classification [16,5]. While some of this work makes trade-offs between efficiency and information disclosure, all maintain provable privacy of individual information and bounds on disclosure, and disclosure is limited to information that is unlikely to be of practical concern.

Privacy preservation can mean many things: Protecting specific individual values, breaking the link between values and the individual they apply to, protecting source, etc. This paper aims for a high standard of privacy: Not only individual entities are protected, but to the extent feasible even the schema (attributes and possible attribute values) are protected from disclosure. Our goal is for each site to disclose as little as possible, while still constructing a valid tree in a time suitable for practical application.

To this end, all that is revealed is the basic structure of the tree (e.g., the number of branches at each node, corresponding to the number of distinct values for an attribute; the depth of each subtree) and which site is responsible for the decision made at each node (i.e., which site possesses the attribute used to make the decision, but not what attribute is used, or even what attributes the site possesses.) This allows for efficient *use* of the tree to classify an object; otherwise using the tree would require a complex cryptographic protocol involving every party at every *possible* level to evaluate the class of an object without revealing who holds the attribute used at that level. Each site also learns the count of classes at some interior nodes (although only the class site knows the mapping to actual classes – other sites don't even know if a class with 30% distribution at one node is the same class as one with a 60% distribution at a lower node, except to the extent that this can be deduced from the tree and it's own attributes.) At the leaf nodes, this is desirable: one often wants probability estimates, not simply a predicted class. As knowing the count of transactions at each leaf node would enable computing distributions throughout the tree anyway, this really doesn't disclose much *new* information.

We now go directly into the algorithm for creating a tree. In Section 3 we describe how the tree (distributed between sites) is used to classify an instance, even though the attribute values of the instance to be classified are also private and distributed between sites. Section 4 formalizes what it means to be secure, and gives a proof that the algorithms presented are secure. Section 5 presents the computation and communication complexity of the algorithm. Section 6 discusses future work and concludes the paper.

2 Privacy-Preserving ID3: Creating the Tree

The basic ID3 algorithm[3] is given in Algorithm 1. We will introduce our distributed privacy-preserving version by running through this algorithm, describing pieces as appropriate. We then give the full algorithm in Algorithm 7. Note

Algorithm 1. ID3(R,C,T) tree learning algorithm

Require: R, the set of attributes
Require: C, the class attribute
Require: T, the set of transactions
if R is empty **then**
  return a leaf node, with class value assigned to most transactions in T
else if all transactions in T have the same class c **then**
  return a leaf node with the class c
else
  Determine the attribute A that best classifies the transactions in T
  Let $a_1, \ldots, a_m$ be the values of attribute A. Partition T into the m partitions $T(a_1), \ldots, T(a_m)$ such that every transaction in $T(a_i)$ has the attribute value a_i.
  Return a tree whose root is labeled A (this is the test attribute) and has m edges labeled $a_1, \ldots, a_m$ such that for every i, the edge a_i goes to the tree $ID3(R - A, C, T(a_i))$.
end if

that for our distributed algorithm, no site knows R, instead each site i knows its own attributes R_i. Only one site knows the class attribute C. In vertical partitioning, every site knows a *projection* of the transactions $\Pi_{R_i} T$. Each projection includes a transaction identifier that serves as a join key.

We first check if R is empty. This is based on Secure Sum[17,10], and is given in Algorithm 2. Basically, the first party adds a random r to its count of remaining items. This is passed to all sites, each adding its count. The last

Algorithm 2. IsREmpty(): Are any attributes left?

Require: k sites P_i (the site calling the function is P_1; any other site can be P_k), each with a flag $AR_i = 0$ if no remaining attributes, $AR_i = 1$ if P_i has attributes remaining.
Require: a commutative encryption function E with domain size $m > k$.
P_1 chooses a random integer r uniformly from $0 \ldots m-1$.
P_1 sends $r + AR_1$ to P_2
for $i = 2..k-1$ **do**
  Site P_i receives r' from P_{i-1}.
  P_i sends $r' + AR_i \bmod m$ to P_{i+1}
end for
Site P_k receives r' from P_{k-1}.
$r' \leftarrow r' + AR_k \bmod m$
P_1 and P_k create secure keyed commutative hash keys E_1 and E_k
P_1 sends $E_1(r)$ to P_k
P_k receives $E_1(r)$ and sends $E_k(E_1(r))$ and $E_k(r')$ to P_1
P_1 returns $E_1(E_k(r')) = E_k(E_1(r))$ $\{\Leftrightarrow r' = r \Leftrightarrow \sum_{j=1}^{k} AR_i = 0 \Leftrightarrow 0$ attributes remain $\}$

site and first then use commutative encryption to compare the final value to r (without revealing either) – if they are the same, R is empty.

Line 2 requires determining the majority class for a node, when only one site knows the class. This is accomplished with a protocol for securely determining the cardinality of set intersection. Many protocols for doing so are known [13,18,19]. We assume that one of these protocols is used. Each site determines which of its transactions *might* reach that node of the tree. The intersection of these sets with the transactions in a particular class gives the number of transactions that reach that point in the tree, enabling the class site to determine the distribution and majority class; it returns a (leaf) node identifier that allows it to map back to this distribution.

To formalize this, we introduce the notion of a *Constraint Set*. As the tree is being built, each party i keeps track of the values of its attributes used to reach that point in the tree in a filter $Constraints_i$. Initially, this is all don't care values ('?'). However, when an attribute A_{ij} at site i is used (lines 6-7 of id3), entry j in $Constraints_i$ is set to the appropriate value before recursing to build the subtree. An example is given in Figure 1. The site has 6 attributes $A_1, \ldots, A_6$. The constraint tuple shows that the only transactions valid for this transaction are those with a value of 5 for A_1, *high* for A_2, and *warm* for A_5. The other attributes have a value of ? since they do not factor into the selection of an instance. Formally, we define the following functions:

A_1	A_2	A_3	A_4	A_5	A_6
5	high	?	?	warm	?

Fig. 1. A constraint tuple for a single site

Constraints.set($attr, val$**):** Set the value of attribute *attr* to *val* in the local constraints set. The special value '?' signifies a don't-care condition.

satisfies: x *satisfies* $Constraints_i$ if and only if the attribute values of the instance are compatible with the constraint tuple: $\forall i, (A_i(x) = v \Leftrightarrow Constraints(A_i) = v) \vee Constraints(A_i) =$ '?'.

FormTransSet: *Function* $FormTransSet(Constraints)$*: Return local transactions meeting constraints*

1: $Y = \emptyset$
2: **for all** transaction id $i \in T$ **do**
3: **if** t_i satisfies *Constraints* **then**
4: $Y \leftarrow Y \cup \{i\}$
5: **end if**
6: **end for**
7: return Y

Now, we determine the majority class (and class distributions) by computing for each class $\bigcap_{i=1..k} Y_i$, where Y_k includes a constraint on the class value. This is given in Algorithm 3.

Algorithm 3. DistributionCounts(): Compute class distribution given current constraints

Require: k sites P_i with local constraint sets $Constraints_i$
1: **for all** sites P_i except P_k **do**
2: at P_i: $Y_i \leftarrow FormTransSet(Constraints_i)$
3: **end for**
4: **for** each class $c_1, \ldots, c_p$ **do**
5: at P_k: $Constraints_k.set(C, c_i)$ {To include the class restriction}
6: at P_k: $Y_k \leftarrow FormTransSet(Constraints_k)$
7: $cnt_i \leftarrow |Y_1 \cap \ldots \cap Y_k|$ using the cardinality of set intersection protocol ([13,18,19])
8: **end for**
9: return $(cnt_1, \ldots, cnt_p)$

The next issue is determining if all transactions have the same class (Algorithm 1 line 3). If all are not the same class, as little information as possible should be disclosed. For efficiency, we do allow the class site to learn the count of classes even if this is an interior node; since it could compute this from the counts at the leaves of the subtree below the node, this discloses no additional information. Algorithm 4 gives the details, it uses constraint sets and secure cardinality of set intersection in basically the manner described above for computing the majority class at a leaf node. If all transactions are in the same class,

Algorithm 4. IsSameClass(): Are all transactions of the same class?

Require: k sites P_i with local constraint sets $Constraints_i$
1: $(cnt_1, \ldots, cnt_p) \leftarrow DistributionCounts()$
2: **if** $\exists j$ s.t. $cnt_j \neq 0 \wedge \forall i \neq j$, $cnt_i = 0$ {only one of the counts is non-zero} **then**
3: Build a leaf node with distribution $(cnt_1, \ldots, cnt_p)$ {Actually, 100% class j}
4: return ID of the constructed node
5: **else**
6: return $false$
7: **end if**

we construct a leaf node. The class site maintains a mapping from the ID of that node to the resulting class distribution.

The next problem is to compute the best attribute: that with the maximum information gain. The information gain when an attribute A is used to partition the data set S is:

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

Algorithm 5. AttribMaxInfoGain(): return the site with the attribute having maximum information gain

```
1: for all sites P_i do
2:    bestgain_i ← −1
3:    for each attribute A_ij at site P_i do
4:       gain ← ComputeInfoGain(A_ij)
5:       if gain > bestgain_i then
6:          bestgain_i ← gain
7:          BestAtt_i ← A_ij
8:       end if
9:    end for
10: end for
11: return argmax_j bestgain_j {Could implement using a set of secure comparisons}
```

Algorithm 6. ComputeInfoGain(A): Compute the Information Gain for attribute A

```
1: S ← DistributionCounts() {Total number of transactions at this node}
2: InfoGain ← Entropy(S)
3: for each attribute value a_i do
4:    Constraints.set(A, a_i) {Update local constraints tuple}
5:    S_{a_i} ← DistributionCounts()
6:    Infogain ← Infogain − Entropy(S_{a_i}) * |S_{a_i}|/|S| {|S| is Σ_{i=1}^{p} cnt_i}
7: end for
8: Constraints.set(A, '?') {Update local constraints tuple}
9: return InfoGain
```

The entropy of a dataset S is given by:

$$Entropy(S) = -\sum_{j=1}^{p} \frac{N_j}{N} \log \frac{N_j}{N}$$

where N_j is the number of transactions having class c_j in S and N is the number of transactions in S. As we see, this again becomes a problem of counting transactions: the number of transactions that reach the node N, the number in each class N_j, and the same two after partitioning with each possible attribute value $v \in A$. Algorithm 6 details the process of computing these counts; Algorithm 5 captures the overall process.

Once the best attribute has been determined, execution proceeds at that site. It creates an interior node for the split, then recurses.

3 Using the Tree

Instance classification proceeds as in the original ID3 algorithm, except that the nodes (and attributes of the database) are distributed. The site requesting classification (e.g., a master site) knows the root node of the classification tree.

Algorithm 7. PPID3(): Privacy-Preserving Distributed ID3

Require: Transaction set T partitioned between sites $P_1, \ldots, P_k$
Require: p class values, $c_1, \ldots, c_p$, with P_k holding the class attribute
1: **if** $IsREmpty()$ **then**
2: Continue at site P_k up to the return:
3: $(cnt_1, \ldots, cnt_p) \leftarrow DistributionCounts()$
4: Build a leaf node with distribution $(cnt_1, \ldots, cnt_p)$
5: $\{class \leftarrow argmax_{i=1..p}\, cnt_i\}$
6: return ID of the constructed node
7: **else if** $clsNode \leftarrow$ (at P_k :) $IsSameClass()$ **then**
8: return leaf nodeId $clsNode$
9: **else**
10: $BestSite \leftarrow AttribMaxInfoGain()$
11: **Continue execution at** $BestSite$**:**
12: Create Interior Node Nd with attribute $Nd.A \leftarrow BestAtt_{BestSite}$ {This is best locally (from $AttribMaxInfoGain()$), and globally from line 8}
13: **for** each attribute value $a_i \in Nd.A$ **do**
14: $Constraints.set(Nd.A, a_i)$ {Update local constraints tuple}
15: $nodeId \leftarrow PPID3()$ {Recurse}
16: $Nd.a_i \leftarrow nodeId$ {Add appropriate branch to interior node}
17: **end for**
18: $Constraints.set(A, \text{'?'})$ {Returning to parent: should no longer filter transactions with A}
19: Store Nd locally keyed by Node ID
20: return Node ID of interior node Nd {Execution continues at site owning parent node}
21: **end if**

The basic idea is that control passes from site to site, based on the decision made. Each site knows the transaction's attribute values for the nodes at its site (and can thus evaluate the branch), but knows nothing of the other attribute values. The complete algorithm is given in Algorithm 8, and is reasonably self-explanatory if viewed in conjunction with Algorithm 7.

We now give a demonstration of how instance classification would actually happen in this instance for the tree built with the UCI "weather" dataset[20]. Assume two sites: The weather observatory collects information about relative humidity and wind, a second collects temperature and cloud cover forecast as well as the class ("Yes" or "No"). Suppose we wish to know if it is a good day to play tennis. Neither sites wants to share their forecasts, but are willing to collaborate to offer a "good tennis day" service. The classification tree is shown in Figure 2, with S1 and S2 corresponding to the site having information on that node. The private information for each site is shown within italics. If today is sunny with normal humidity, high temperature, and weak wind; classification would proceed as follows: We know that Site 1 has the root node (we don't need to know anything else). Site 1 retrieves the attribute for from $S1L1$: Outlook. Since the classifying attribute is outlook, and Site 1 knows the forecast is sunny,

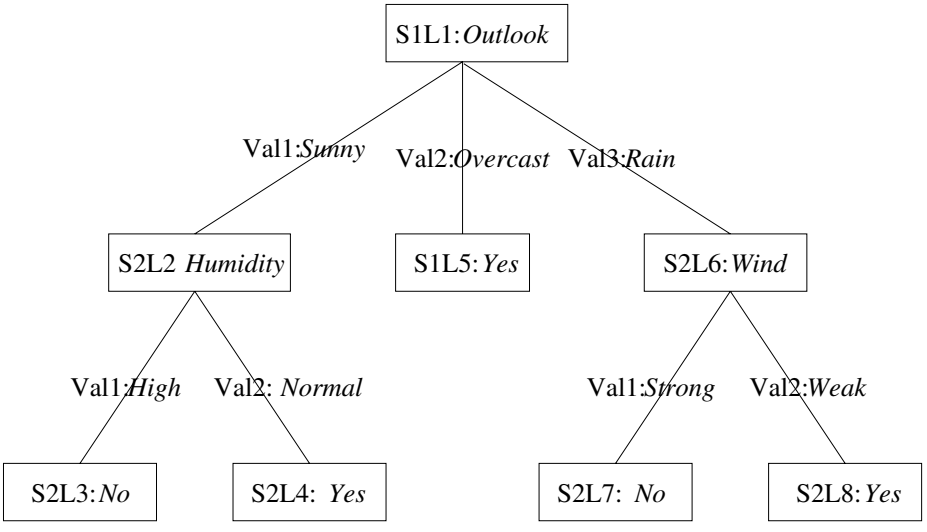

Fig. 2. The privacy preserving ID3 decision tree on the weather dataset (Mapping from identifiers to attributes and values is known only at the site holding attributes)

the token $S2L2$ is retrieved. This indicates that the next step is at Site 2. Site 2 is called with the token $S2L2$, and retrieves the attribute for $S2L2$: Humidity. The humidity forecast is normal, so the token $S2L4$ is retrieved. Since this token is also present at Site 2, it retrieves the class value for nodeId $S2L4$ and returns it: we receive our answer of "Yes".

4 Security Discussion

We evaluate the security of our algorithm under the basic framework of Secure Multiparty Computation [21]. As such, we assume the security of the underlying

Algorithm 8. classifyInstance(instId, nodeId): returns the class/distribution for the instance represented by instId

1: {The start site and ID of the root node is known}
2: **if** $nodeId$ is a LeafNode **then**
3: return class/distribution saved in $nodeId$
4: **else** {$nodeId$ is an interior node}
5: $Nd \leftarrow$ local node with id $nodeId$
6: $value \leftarrow$ the value of attribute $Nd.A$ for transaction $instId$
7: $childId \leftarrow Nd.value$
8: return $childId.Site.classifyInstance(instId, childId)$ {Actually tail recursion: this site need never learn the class}
9: **end if**

set intersection algorithm, and then prove the security of our privacy-preserving ID3 algorithm.

The proof of security is given assuming semi-honest adversaries. A semi-honest party follows the rules of the protocol using its correct input, but is free to later use what it sees during execution of the protocol to compromise security. While this protocol provides somewhat strong guarantees in the absence of collusion, due to space constraints we will only prove security for the semi-honest case.

Privacy by Simulation. The basic proof style is to show that the view of each party during the execution of the protocol can be effectively simulated given the input and the output of that party. This is sufficient to prove that the protocol is secure [21]. Thus, in all of the following proofs of security, we show that we can simulate each message received. Once the received messages are simulated, the algorithm itself can be used to simulate the rest of the view. This does not quite guarantee that private information is protected. Whatever information can be deduced from the final result is not kept private. However, nothing *beyond* the results is learned.

4.1 Secure ID3

We first analyze the security of the constituent algorithms, then the security of the complete algorithm. Although it may seem that some of the constituent algorithms leak a large quantity of information, in the context of the full algorithm the leaked information can be simulated by knowing the distribution counts at each node, so overall privacy is maintained.

Lemma 1. *Algorithm 2 reveals nothing to any site except whether the total number of attributes left is* 0*.*

Proof. The algorithm has two basic phases: The sum (through P_k), and the comparison between P_k and P_1. First, the sum: simulating the messages received at lines 2 and 7. The value received by P_i at these steps is $r + \sum_{j=1}^{i-1} AR_j \bmod m$. We will simulate by choosing a random integer uniformly from $0 \ldots m-1$ for r'. We now show that the probability that the simulated $r' = x$ is the same as the probability that the messages received in the view $= x$.

$$\begin{aligned} Pr\{VIEW_i = x\} &= Pr\{x = r + \sum_{j=1}^{i-1} AR_j \bmod m\} \\ &= Pr\{r = x - \sum_{j=1}^{i-1} AR_j \bmod m\} \\ &= \frac{1}{m} \\ &= Pr\{Simulator_i r' = x\} \end{aligned}$$

The key to the derivation is that arithmetic is mod m. r and r' are chosen uniformly from $0 \ldots m-1$, so the probability of hitting any particular value in that range is $1/m$.

Simulating the message received by P_k at line 11 is simple: Secure encryption gives messages where the distribution is independent of the key/message, so a selection from this distribution of possible encrypted messages simulates what P_k receives.

The messages received by P_1 are more difficult. The problem is that if $r = r'$, $E_k(r')$ must be such that when encrypted with E_1 it is equal to $E_k(E_1(r))$. For this, the simulator requires the ability to decrypt. The simulator computes $m = D_1(E_k(E_1(r)) = E_k(r)$. If $r = r'$, this is the message used to simulate $E_k(r')$. If not, a random message $\neq m$ is chosen, as in the simulator for P_k. □

Lemma 2. *Algorithm 3 reveals only the count of instances corresponding to all combinations of constraint sets for each class.*

Proof. The only communication occurs at line 7 which consists of a call to the Cardinality of Set Intersection algorithm. This reveals only the size of the intersection set for all subsets of Y_i, which are the counts revealed. Algorithm 3 is secure except for revealing this information. □

Lemma 3. *Algorithm 4 finds if all transactions have the same class, revealing only the class distributions described in Lemma 2.*

Proof. Line 1 is an invocation of Algorithm 3; Everything else is computed locally, and can be simulated from the knowledge from Lemma 2. □

Lemma 4. *Algorithm 6 reveals nothing except the counts S, S_{a_i}, and the constituent subcounts described in Lemma 2 for each attribute value a_i and class j, assuming the number of distinct class values is known.*

Proof. The only messages received are at lines 1 and 5, invocations of the *DistributionCounts*() function. Since the underlying function is secure, Algorithm 6 is secure. □

Lemma 5. *Algorithm 5 finds the site with the attribute having the maximum information gain while revealing only the best information gain at each site and the information discussed in Lemma 4.*

Proof. Communication occurs at lines 4 and 11. Line 4 consists of an invocation of Algorithm 6. Line 11 is implemented by letting the site compare all the values; revealing the value of the best information gain at each site. Assuming this is revealed (part of the input to the simulator), it is trivially simulated. □

Further reduction of the information revealed is possible by using a secure protocol for finding the maximum among a set of numbers. This would reveal only the site having the attribute with the maximum information gain and nothing else.

Theorem 1. *Algorithm 7 computes the decision tree while revealing only:*

- *The distribution subcounts of each node, as described in Lemma 2. (The full counts, and some of the subcounts, can be computed knowing the distribution counts at the leaves.)*
- *The best information gain from each site at each interior node (as discussed above, this leak can be reduced.)*

Proof. Knowing the final tree, the simulator at each site can uniquely determine the sequence of node computations at a site and list the function calls occurring due to this. Given this function call list, if the messages received in each function call can be simulated, the entire algorithm can be proven to be secure.

Line 1 is an invocation of Algorithm 2. The result is simulated as either true or false depending on whether the node in question is a leaf node in the final tree or not.

Line 3 is an invocation of Algorithm 3. The actual counts are given by the counts *in* the leaf node, which are known to the site P_k that invoked the algorithm. The subcounts revealed by Algorithm 3 are presumed known.

Line 7 is an invocation of Algorithm 4. If the node in question is not a leaf node in the final tree, the result is false. Otherwise the result is the nodeId of the leaf node.

Line 10 consists of an invocation of Algorithm 5. The result is actually equal to the Site which will own the child node. This information is known from the tree structure. The subcounts and information gain values revealed during this step are presumed known.

Line 15 is a recursive invocation that returns a node identifier; a part of the tree structure.

Since all of the algorithms mentioned above have been proven secure, applying the composition theorem, Algorithm 7 is secure. The repeated invocations of the cardinality of set intersection protocol are valid because in each invocation, a new set of keys are chosen. This ensures that messages cannot be correlated across calls. □

Theorem 2. *Algorithm 8 reveals nothing other than the leaf node classifying the instance.*

Proof. All the computations are local. The only information passed between various sites are node identifiers. This list of node identifiers can be easily simulated from the classification tree once the final leaf is known. □

5 Computation and Communication Analysis

The communication/computation analysis depends on the number of transactions, number of parties, number of attributes, number of attribute values per attribute, number of classes and complexity of the tree. Assume that there are: n transactions, k parties, c classes, r attributes, p values per attribute (on average), and q nodes in final classification tree. We now give a rough analysis of the

cost involved in terms of the number of set intersections required for building the tree (erring on the conservative side).

At each node in the tree the best classifying attribute needs to be determined. To do this, the entropy of the node needs to be computed as well as the information gain per attribute. Computing the entropy of the node requires c set intersections (1 per class). Computing the gain of one attribute requires cp set intersections (1 per attribute value and class). Thus, finding the best attribute requires cpr set intersections. Note that this analysis is rough and assumes that the number of attributes available at each node remains constant. In actuality, this number linearly decreases with the depth of the node in the tree (this has little effect on our analysis). In total, every node requires $c(1+pr)$ set intersections. Therefore, the total tree requires $cq(1+pr)$ set intersections.

The intersection protocol of [13] requires that the set of each party be encrypted by every other party. Since there are k parties, k^2 encryptions are required and k^2 sets are transferred. Since each set can have at most n transactions, the upper bound on computation is $O(nk^2)$ and the upper bound on communication cost is also $O(nk^2 * bitsize)$ bits.

Therefore, in total the entire classification process will require $O(cqnk^2(1+pr))$ encryptions and $cqnk^2(1+pr) * bitsize$ bits communication. Note that the encryption process can be completely parallelized reducing the required time by an order of k.

Once the tree is built, classifying an instance requires no extra overhead, and is comparable to the original $ID3$ algorithm.

6 Conclusions

It is possible to extend the protocols developed such that the class of each instance is learned only by the party holding the class attribute (nothing is learned by the remaining parties). In some cases, this might be preferable.

The major contributions of this paper are the following:

- It proposes a new protocol to construct a decision tree on vertically partitioned data with an arbitrary number of parties where only one party has the class attribute (The method is trivially extendible to the case where all parties have the class attribute, and in fact causes a significant increase in the efficiency of the protocol).
- The paper presents a general framework in which distributed classification would work and how such a system should be constructed.

As part of future work, we are actually implementing the entire protocol in JAVA, which should form the first working code in the area of PPDM. Our work provides an upper bound on the complexity of building privacy preserving decision trees. Significant work is required to propose more efficient solutions and/or to find a tight upper bound on the complexity. We leave this for the future.

References

1. Agrawal, R., Srikant, R.: Privacy-preserving data mining. In: Proceedings of the 2000 ACM SIGMOD Conference on Management of Data, Dallas, TX, ACM (2000) 439–450
2. Lindell, Y., Pinkas, B.: Privacy preserving data mining. In: Advances in Cryptology – CRYPTO 2000, Springer-Verlag (2000) 36–54
3. Quinlan, J.R.: Induction of decision trees. Machine Learning **1** (1986) 81–106
4. Lindell, Y., Pinkas, B.: Privacy preserving data mining. Journal of Cryptology **15** (2002) 177–206
5. Du, W., Zhan, Z.: Building decision tree classifier on private data. In Clifton, C., Estivill-Castro, V., eds.: IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining. Volume 14., Maebashi City, Japan, Australian Computer Society (2002) 1–8
6. Agrawal, D., Aggarwal, C.C.: On the design and quantification of privacy preserving data mining algorithms. In: Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Santa Barbara, California, USA, ACM (2001) 247–255
7. Evfimievski, A., Srikant, R., Agrawal, R., Gehrke, J.: Privacy preserving mining of association rules. In: The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada (2002) 217–228
8. Rizvi, S.J., Haritsa, J.R.: Maintaining data privacy in association rule mining. In: Proceedings of 28th International Conference on Very Large Data Bases, Hong Kong, VLDB (2002) 682–693
9. Kargupta, H., Datta, S., Wang, Q., Sivakumar, K.: On the privacy preserving properties of random data perturbation techniques. In: Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03), Melbourne, Florida (2003)
10. Kantarcıoğlu, M., Clifton, C.: Privacy-preserving distributed mining of association rules on horizontally partitioned data. IEEE Transactions on Knowledge and Data Engineering **16** (2004) 1026–1037
11. Rozenberg, B., Gudes, E.: Privacy preserving frequent item-set mining in vertically partitioned databases. In: Proceedings of the Seventeenth Annual IFIP WG 11.3 Working Conference on Data and Applications Security, Estes Park, Colorado, U.S.A. (2003)
12. Vaidya, J., Clifton, C.: Privacy preserving association rule mining in vertically partitioned data. In: The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada (2002) 639–644
13. Vaidya, J., Clifton, C.: Secure set intersection cardinality with application to association rule mining. Journal of Computer Security (to appear)
14. Lin, X., Clifton, C., Zhu, M.: Privacy preserving clustering with distributed EM mixture modeling. Knowledge and Information Systems (to appear 2004)
15. Vaidya, J., Clifton, C.: Privacy-preserving k-means clustering over vertically partitioned data. In: The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC (2003) 206–215
16. Vaidya, J., Clifton, C.: Privacy preserving naïve bayes classifier for vertically partitioned data. In: 2004 SIAM International Conference on Data Mining, Lake Buena Vista, Florida (2004) 522–526

17. Schneier, B.: Applied Cryptography. 2nd edn. John Wiley & Sons (1995)
18. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Eurocrypt 2004, Interlaken, Switzerland, International Association for Cryptologic Research (IACR) (2004)
19. Agrawal, R., Evfimievski, A., Srikant, R.: Information sharing across private databases. In: Proceedings of ACM SIGMOD International Conference on Management of Data, San Diego, California (2003)
20. Blake, C., Merz, C.: UCI repository of machine learning databases (1998)
21. Goldreich, O.: General Cryptographic Protocols. In: The Foundations of Cryptography. Volume 2. Cambridge University Press (2004)