A Random Decision Tree Framework for Privacy-Preserving Data Mining

Jaideep Vaidya, *Senior Member, IEEE*, Basit Shafiq, *Member, IEEE*, Wei Fan, *Member, IEEE*, Danish Mehmood, and David Lorenzi

Abstract—Distributed data is ubiquitous in modern information driven applications. With multiple sources of data, the natural challenge is to determine how to collaborate effectively across proprietary organizational boundaries while maximizing the utility of collected information. Since using only local data gives suboptimal utility, techniques for privacy-preserving collaborative knowledge discovery must be developed. Existing cryptography-based work for privacy-preserving data mining is still too slow to be effective for large scale data sets to face today's big data challenge. Previous work on random decision trees (RDT) shows that it is possible to generate equivalent and accurate models with much smaller cost. We exploit the fact that RDTs can naturally fit into a parallel and fully distributed architecture, and develop protocols to implement privacy-preserving RDTs that enable general and efficient distributed privacy-preserving knowledge discovery.

Index Terms—Privacy-preserving data mining, classification

1 Introduction

Building and applying any data mining model generally assumes that the underlying data is freely accessible. Often, this is not realistic. Privacy and security concerns restrict the sharing or centralization of data. Privacy-preserving data mining has emerged as an effective method to solve this problem [1]. Distributed solutions have been proposed that can preserve privacy while still enabling data mining. However, while perturbation based solutions do not provide stringent privacy, cryptographic solutions are too inefficient and infeasible to enable truly large scale analytics to face the era of big data. In this paper, we propose a solution that uses both randomization and cryptographic techniques to provide improved efficiency and security for several decision tree-based learning tasks. Indeed, to the best of our knowledge, the proposed solution provides an order of magnitude improvement in efficiency over existing solutions while providing more security. This is an effective solution to privacy-preserving data mining for the big data challenge.

The proposed approach is based on random decision trees (RDT), developed by Fan et al. [2]. One important property of RDT is that the same code can be used for multiple data mining tasks: classification, regression, ranking and multiple classification [2], [3], [4]. As shown previously, the RDT is an efficient implementation of Bayes optimal classifier (BOC) [2], effective non-parametric density estimation [4], and can be explained via high order statistics such as moments [5]. The use of the multiple RDTs in various learning tasks offers many benefits over other traditional classification/tree building techniques, because its structure and progression lends itself to modification for distributed/parallel tasks. RDT is also an excellent candidate for use in privacy preserving distributed data mining since:

1. Randomness in structure rather than simple perturbation of input/output is more effective—perturbing the input or output from a database to achieve privacy works, but the utility of the information garnered from data mining can be diminished if the perturbations are not carefully controlled, or conversely, information can be leaked if the information is not perturbed enough. Instead, we can exploit the design properties of RDT to generate trees that are random in structure, providing us with a similar end effect as perturbation without the associated pitfalls. A random structure provides security against leveraging a priori information to discover the entire classification model or instances.
2. Purely cryptographic approaches are often too slow to be practical and can become computationally expensive as the size of the data set increases and intercommunications between different parties increase. RDT provides a convenient escape from this paradigm thanks to its structural properties, more specifically, the fact that only specific nodes (the leaves) in the classification tree need to be encrypted/decrypted, and secure token passing prevents attackers from utilizing counting techniques to decipher instance classifications, as the branch structure of the tree is hidden from all parties.
3. RDT is a general approach in which the same code works for classification, regression, ranking

- *J. Vaidya and D. Lorenzi are with the MSIS Department, Rutgers University, 1 Washington Park, Newark, NJ 07102.*
E-mail: jsvaidya@business.rutgers.edu, dlorenzi@pegasus.rutgers.edu.
- *B. Shafiq and D. Mehmood are with the CS Department, Lahore University of Management Sciences, D.H.A., Lahore Cantt., Lahore 54792, Pakistan. E-mail: basit@lums.edu.pk, dmehmood@gmail.com.*
- *W. Fan is with Huawei Noah's Ark Lab, Core Building 2, Hong Kong Science Park, Shatin, Hong Kong. E-mail: wei.fan@gmail.com.*

Manuscript received 22 Dec. 2012; revised 6 July 2013; accepted 22 Sept. 2013. Date of publication 30 Sept. 2013; date of current version 17 Sept. 2014.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TDSC.2013.43

and multi-label classification. Thus with the same techniques, we can solve four typical learning problems in the same framework.

4. An additional security advantage of RDTs is that they can be very easily made differentially private. As shown by Jagannathan et al. [6], the node statistics can be viewed as queries over the training data. Therefore, standard techniques can be used to return differentially private results, without significant loss of accuracy.

In this paper, we develop methods to securely construct RDTs for both horizontally and vertically partitioned data sets. We implement the proposed protocols and analyze the computation and communication cost, and security. We also compare the performance of the proposed protocols with the existing ID3-based protocols [7].Our main contribution is to realize that RDTs can provide good security with very high efficiency.

2 Random Decision Trees

While the use of RDTs may seem counterintuitive, there are many benefits in terms of performance and accuracy, that are gained by using this method versus traditional algorithms. Fan et al. [2] find that for classification, use of a random model can match, in terms of solutions, other inductive learning models in finding an optimal hypothesis. At the same time, RDT outperforms other models in terms of computational speed, due to the inherent properties of random partitioning used in tree construction.

The RDTs algorithm builds multiple (or m) iso-depth RDTs. One important aspect of RDTs is that the structure of a random tree is constructed completely independent of the training data. The RDT algorithm can be broken into two stages, training and classification. The training phase consists of building the trees (BuildTreeStructure) and populating the nodes with training instance data (UpdateStatistics). It is assumed that the number of attributes is known based on the training data set. The depth of each tree is decided based on a heuristic—Fan et al. [2] show that when the depth of the tree is equal to half of the total number of features present in the data, the most diversity is achieved, preserving the advantage of random modeling.

The process for generating a tree is as follows. First, start with a list of features (attributes) from the data set. Generate a tree by randomly choosing one of the features without using any training data. The tree stops growing once the height limit is reached. Then, use the training data to update the statistics of each node. Note that only the leaf nodes need to record the number of examples of different classes that are classified through the nodes in the tree. The training data is scanned exactly once to update the statistics in multiple random trees. When classifying a new instance x, the probability outputs (or regression/ranking values for regression, ranking and multi-label classification problems) from multiple trees are averaged to estimate the a posteriori probability.

2.1 Running Example

Table 1 shows the well known weather data set distributed between two parties. We assume that when the data set is horizontally partitioned, instances 1-7 are owned by Party 1, while 8-14 are owned by Party 2. If it is vertically partitioned, we assume that Party 1 owns the outlook and temperature attributes while Party 2 owns the humidity, windy, and play attributes. To save space, both cases are shown together in Table 1. For simplicity, we assume that only two RDT trees are built (depicted in Fig. 1). Suppose a new instance {sunny,mild,normal,weak} is to be classified. Then, as per the first random tree, the prediction is (2, 0) without normalization. The prediction as per the second random tree is (1, 2). Therefore, the non-normalized overall class distribution vector provided by RDT is (1.5, 1).

TABLE 1
The Distributed Weather Data Set

	—P1—		—P2—		
	outlook	temperature	humidity	windy	play
P1	sunny	hot	high	weak	no
	sunny	hot	high	strong	no
	overcast	hot	high	weak	yes
	rainy	mild	high	weak	yes
	rainy	cool	normal	weak	yes
	rainy	cool	normal	strong	no
	overcast	cool	normal	strong	yes
P2	sunny	mild	high	weak	no
	sunny	cool	normal	weak	yes
	rainy	mild	normal	weak	yes
	sunny	mild	normal	strong	yes
	overcast	mild	high	strong	yes
	overcast	hot	normal	weak	yes
	rainy	mild	high	strong	no

3 Problem Statement

Since the same RDT code can be used for multiple data mining tasks, we focus on classification for ease of discussion. The basic problem in distributed classification is to train a classifier from the distributed data and then classify each new instance. For distributed decision tree classification, the objective is to create a decision tree classifier from the distributed data. In the privacy-preserving case, the additional constraint is that the process of building the classifier, or of classifying an instance should not leak any additional information beyond what is learned from the result (and the local input). Assuming the global data set $D \equiv (T, R)$, where T represents the global set of transactions, and R represents the global schema, the general problem can be formulated as follows.

Definition 1 (Privacy-Preserving RDT). *Given a data set $D \equiv (T, R)$ distributed among k parties $P_1, \ldots, P_k$, securely build a random decision tree classifier RDT, and provide a privacy-preserving distributed classification mechanism to classify a new instance.*

Clearly, the specific steps are dependent on the ways in which the data is distributed. The two most common are horizontal and vertical partitions. When data is horizontally partitioned between k parties, each party holds different instances, but collects the same pieces of information. All parties share the schema, though the specific transactions in their local databases are unique. In this case, we assume that there are no overlapping transactions. Clearly, since the schema is shared by all parties, the class attribute C is also known to all parties. An example of this would be the case of banks collecting credit card transaction records. They

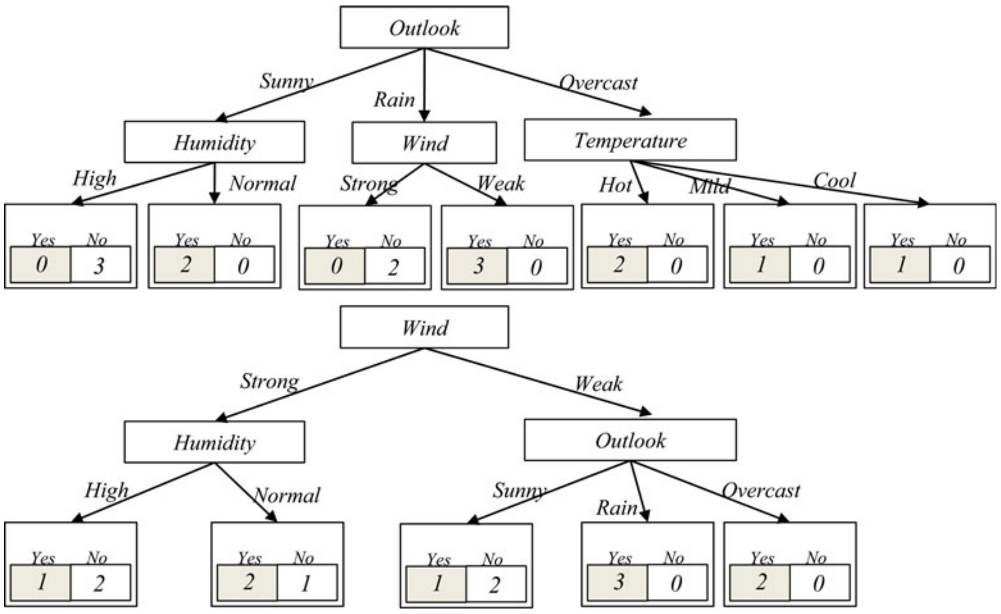

Fig. 1. Two random decision trees for the weather data set.

collect the same information, though the specific transactions are unique to the credit card, and therefore do not overlap.

In the case of vertical partitioning, parties collect different information about the same set of entities. The class attribute is known to only one site, which is the one that owns the class attribute. Note that though the class attribute could be known to all parties, this is the more general case, and is therefore considered. For instance, a bank, health insurance company and auto insurance company collect different information about the same group of people. A bank has customer information like average monthly deposit, account balance, etc. The health insurance company has access to medical information and other policy information. The car insurance company has access to information such as car type, accident claims, etc. Together, they can provide a reliable model to evaluate if the person is a credit risk for life insurance. We do not address the case of arbitrarily partitioned data, and leave it for future work.

Adversary model. Our methods are robust to semi-honest adversaries [8] (who act according to their prescribed actions in the protocol, but may try to learn more about the other participants). Our protocols are robust to collusion between participants. While our methods are not robust to fully malicious adversaries (who can behave arbitrarily), there are general techniques to transform any protocol valid in the semi-honest model to the fully malicious model (at the cost of efficiency) [8]. The security definition is based on a comparison between an ideal implementation and a trusted third party, where in a real implementation of the protocol (without the trusted third party) no additional information is leaked beyond that in the ideal implementation, with overwhelming probability.

4 Horizontally Partitioned Data

When data is horizontally partitioned, parties collect data for different entities, but have data for all of the attributes. We now need to figure out how the RDTs can be constructed and how classification is performed. Since all the parties share the schema, a straight-forward solution is for all parties to independently create a few random trees. Together these will form the ensemble of random trees. However, each party can only independently create the structure of the tree. All parties must co-operatively and securely compute the parameters (i.e., values of each leaf node), over the global data set. Unlike the basic RDT approach, there is no need to keep the class distribution at each non-leaf node—this information is only required at the leaf nodes. Now, there are two possibilities: 1) The structure of the tree is known to each participant. 2) The structure of the tree is unknown to each participant.

Within case (1) there are three further possibilities: (1a) The global class distribution vector for each leaf node will be known to all parties. (1b) The global class distribution vector for each leaf node is known only to the party owning the tree. (1c) The global class distribution vector for each leaf node is known to none of the parties. While, for case (2) there are two further possibilities: (2a) The values for each leaf node is known to the party owning the tree. (2b) The values for each leaf node is known to none of the parties.

It is obvious that case (2) is inherently more difficult than case (1), as the structure of the tree is unknown to all of the parties. This causes a problem since the other parties can no longer compute the local leaf values. Every party has to collaborate with the tree owner in some way to find out the leaf node values. However, it also does not make too much sense in this context. First, as the schema is known to everyone, and the tree structure is random, anyone could come up with that particular structure. Indeed, it would actually be better to weed out random structures that may be unacceptable to some parties due to privacy concerns. Secondly, even if the structure is unknown, every classification of a new instance can reveal some knowledge about the tree. For example, if a number of new instances are classified to the same leaf node, it is easy to figure out (or at least narrow down) what is the structure of the branch leading to that leaf. Finally, even though this may be possible from the secure computation perspective, it is likely to negate the efficiency advantage of RDTs, thus eliminating the main reason for choosing the random tree approach in the first place. For these reasons just discussed, we do not consider case (2) in the rest of the paper, and only discuss the subcases of case (1) below.

4.1 Case 1a

Here, each party knows the structure for all of the trees. Indeed, if a party thinks that a tree reveals too much information (i.e., one of its paths lead to a case where the leaf node only has a single instance), it might wish to reject that particular tree and ask for alternatives. This can be achieved using an electronic voting protocol [9] if it is necessary to ensure that no participant learns which other participant(s) rejected any tree. Note, however, that rejecting a tree itself leaks information. Even if it is not known who rejected the tree, any party who does not have an issue with the tree knows that at least one other party was not satisfied with the tree. In the worst case, if there are only 2 parties, one party will know that the tree causes privacy breach for the other party. However, it is not known which path of the tree may lead to privacy leakage. In any case, this is unavoidable since a party can either accept a tree or reject it (and accepting in such cases may lead to larger leakage).

Once all parties agree on all of the trees, the computation of the leaf values proceeds. To do this, all parties locally compute the values for the leaf nodes. Following this, they simply have to add up all of the leaf values to get the global tree values. This can be easily done in a secure fashion using the secure sum protocol [10]. Since all parties have the information for all of the trees, classification is simple and can be locally performed. Thus, this case requires no interaction for classifying any new instance.

4.2 Case 1b

Here, the secure sum is run by the party owning the tree, and the results are not distributed to all of the other parties.

Classification would now require interaction between the parties. When a party P_i wishes to classify a new instance, it generates a public-private keypair for a threshold additively homomorphic public-key cryptosystem, such as threshold Damgard-Jurik encryption [11]. Additively homomorphic encryption has the property that $E(m1) * E(m2) = E(m1 + m2)$ (i.e., it is possible to get the encryption of the sum of two values by operating over the individual encrypted values). Additionally, threshold encryption has the property that while anyone can encrypt, decryption requires interaction between the threshold number of participants. P_i then sends the public key to all of the parties, which encrypt the leafs of all of the trees with that public key and send the encrypted trees back to P_i. P_i can now retrieve from each tree the encrypted class distribution vector corresponding to the leaf node that the new instance reaches in that tree. It would then multiply all retrieved encrypted vectors (giving the sum due to the homomorphism), and go through the threshold decryption to retrieve the classification result.

4.3 Case 1c

This case is the most challenging. Here, the secure sum should result in either all parties having a share of the sum, or the party owning the tree having an encrypted vector for the sum that it cannot decrypt. The final classification needs to take this into account to figure out what is the actual class distribution. We again use threshold additively homomorphic encryption to achieve this. Thus, each party first creates a set of random trees and communicates the structure to the other parties. This is repeated until all parties agree to the trees (we assume that a secure electronic voting [9] protocol could be used if it is necessary to conceal which party objects to a particular tree). Now, all parties locally compute the class distributions for the leaf nodes for all of the trees. These are then encrypted using the public key for the additively homomorphic threshold encryption system and sent to all of the other parties. Every party multiplies the encrypted vector components for each leaf node thus giving the encrypted sum. This is due to the additive homomorphic property. However, this cannot be decrypted without the aid of the other parties for the use of threshold encryption, and thus, effectively, each party has all of the global trees in encrypted form. Algorithm 1 gives the details.

Algorithm 1 Building the random trees for horizontally partitioned data

```
Require: Transaction set $T$ partitioned horizontally between sites $P_1, \dots, P_k$
Require: $n_i$, the number of random trees to be created by each participant, such that $\sum_i n_i = m$, the total number of random trees
1: while Every party does not agree to all of the random trees do
2:     {A secure electronic voting protocol can be used if no party should learn which party objects to any of the trees}
3:     Each party generates its random trees
4:     The structure of every tree is communicated to all of the parties
5: end while
6: for each tree $T_j$ do
7:     Each party $P_i$ locally computes the class distribution vectors for each leaf node in $T_j$
8:     Each party $P_i$ encrypts the class distribution vectors for all leaf nodes in $T_j$ using the threshold additively homomorphic encryption system and sends to all other parties
9:     All parties then multiply the corresponding encrypted class distribution vector elements they receive for each leaf node to get the encrypted global value for that node
10: end for
```

When a new instance needs to be classified, the party owning the instance identifies all of the leaf nodes that it reaches, and multiplies the encrypted class vector components together to get the encrypted sum of the class distribution vectors as per each tree. This is now collaboratively decrypted and averaged to get the actual class distribution vector. Note that getting the sum does not reveal more than getting the average since the sum can always be retrieved from the average and the total number of trees. Algorithm 2 gives the specific steps.

Fig. 2 shows the random trees locally updated at each party. At this point, each party would share the

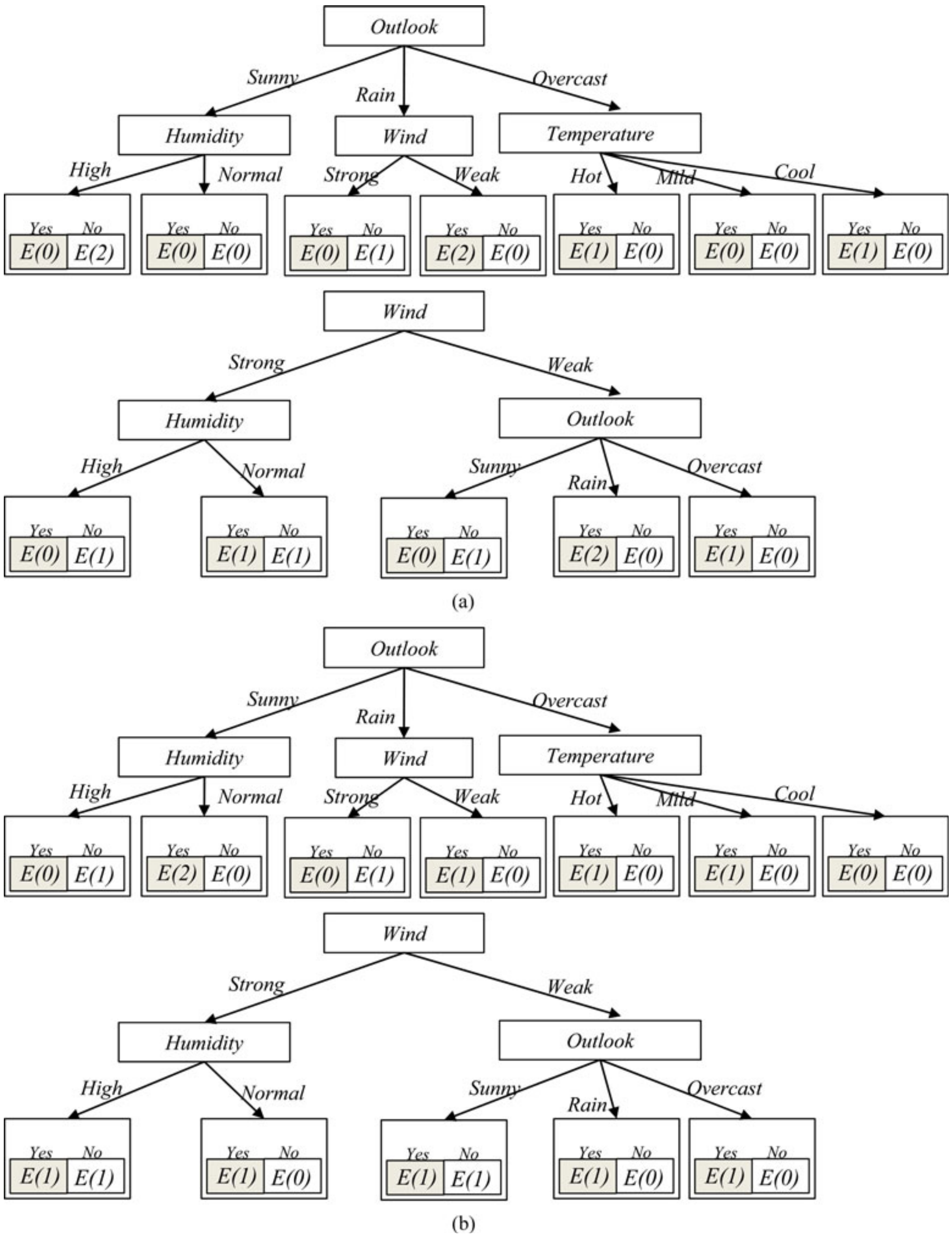

Fig. 2. The random trees built at each party for horizontal partitioning before aggregation. a) Random trees at party 1. b) Random trees at party 2.

encrypted leaf vectors with the other party which would homomorphically add them to get the encrypted leaf vectors corresponding to the trees in Fig. 1. Classification of a new instance is straightforward. Once the encrypted leaf vectors corresponding to that instance have been found, and homomorphically added, threshold decryption is undertaken to get the actual values.

5 Vertically Partitioned Data

With vertically partitioned data, all parties collect data for the same set of entities. However, each party collects data for a different set of attributes. Now the parties cannot independently create even the structure of a random tree, unless they share the attribute information among each other. Thus, there are two possibilities:

- All parties share basic attribute information (i.e., metadata). Now they can independently create random trees (at least the structure).
- There is no sharing of information. Now, the parties need to collaborate to create the random trees. These trees could themselves exist in a distributed form.

Unlike the horizontal partitioning case, the structure of the tree does reveal potentially sensitive information, since the parties do *not* know what are the attributes owned by the other parties. Therefore, we directly address the case of fully distributed trees.

5.1 Fully Distributed Trees

We assume that every site knows the total number of random trees, denoted by m. The algorithms for creating random trees is given in Algorithm 3 that invokes `BuildTree` Algorithm 4 for m times, in order to build m random trees of depth $n/2$, where n is the total number of attributes in the global schema R and is computed by all sites together using the secure sum protocol (line 1 of Algorithm 3). As discussed in [2], with the total number of random trees to be 30 and each of depth $n/2$ (i.e., depth is half of the total number of attributes), the average probability that the random tree has exactly the same classification accuracy as the single best tree (e.g., ID3) is over 95 percent. Therefore, we fix the depth of the random tree to be $n/2$ in Algorithm 3.

Algorithm 2 classifyInstanceHoriz(instId)

```
Require: $m$, the number of random trees built (let $nodeid_j$ represent the root node of the $j$th tree)
1: for $i = 1 \ldots p$ do
2:   Randomly choose r from the range of random numbers for the cryptographic system
3:   $lv_i \leftarrow Encrypt(0, r)$ {Random encryption of 0}
4: end for
5: for $j = 1 \ldots m$ do
6:   $currstats \leftarrow$ encrypted $classdistributionvector$ for instance $instId$ as per tree $j$
7:   for $i = 1 \ldots p$ do
8:     $lv_i \leftarrow lv_i * currstats_i$
9:   end for
10: end for
11: Collaboratively decrypt each $lv_i$ and divide by $m$ to get actual statistics
```

Algorithm 3 CreateRandomTrees

```
Require: Transaction set $T$ partitioned vertically between sites $P_1, \ldots, P_k$
Require: $P_i$ holds $n_i$ attributes
Require: $p$ class values, $c_1, \ldots, c_p$, with $P_k$ holding the class attribute
Require: $m$, the number of random trees to build
1: All parties together compute $n = \sum_i n_i$ using the secure sum protocol [10]
2: $depth \leftarrow n/2$ {The depth of the random trees}
3: for $i = 1 \ldots m$ {Build the ith tree} do
4:   $level \leftarrow 1$
5:   $nodeId_i \leftarrow BuildTree(level, depth)$
6: end for
```

Each random tree is split among the different sites and is built in a distributed fashion using the recursive `BuildTree` procedure given in Algorithm 4. This algorithm builds the structure of a random decision tree without the transaction set. Once the structure is constructed, the statistics of each node is updated by scanning the training example one by one in a distributed and privacy-preserving manner. To give some details, at first, `BuildTree` is invoked by the `primary` site, which initiates creation of each of the m trees and stores pointers to the root node of all these trees. The `BuildTree` algorithms generates a random tree in a recursive manner and in each recursion the $level$ of the current tree is incremented by 1. The recursion ends when the $level$ of the current tree equals to the $depth$ which is fixed in the `CreateRandomTrees` algorithm. In the initial call to the `BuildTree`, first a site j is randomly selected which then randomly selects an attribute (other than the class attribute) from its attribute set (lines 1-2). A node is created corresponding to the selected attribute. This node serves as the root node of the random tree, stored at site j.

Algorithm 4 BuildTree(level,depth)

```
1: if $level \leq depth$ then
2:   Randomly choose $j$ from $[1 \ldots k]$ (uniform random distribution)
3:   At $P_j$: Randomly choose $r$ from $[1 \ldots n_j]$ (uniform random distribution)
4:   At $P_j$: Create Interior Node $Nd$ with attribute $Nd.A \leftarrow A_r$ (the rth attribute)
5:   if $A_r$ is numeric then
6:     At $P_j$: Randomly choose a split point $\pi$ from within the range of $A_r$
7:     At $P_j$: $Constraints.set(Nd.A_r, \text{"} <\pi\text{"})$ {Update local constraints tuple}
8:     $nodeId \leftarrow BuildTree(level + 1, depth)$
9:     $Nd.leftsubtree \leftarrow nodeId$ {Add appropriate branch to interior node}
10:    At $P_j$: $Constraints.set(Nd.A_r, \text{"} >= \pi\text{"})$ {Update local constraints tuple}
11:    $nodeId \leftarrow BuildTree(level + 1, depth)$
12:    $Nd.rightsubtree \leftarrow nodeId$ {Add appropriate branch to interior node}
13:  else
14:    for each attribute value $a_i \in Nd.A$ do
15:      $Constraints.set(Nd.A_r, a_i)$ {Update local constraints tuple}
16:      $nodeId \leftarrow BuildTree(level + 1, depth)$
17:      $Nd.a_i \leftarrow nodeId$ {Add appropriate branch to interior node}
18:    end for
19:  end if
20:  $Constraints.set(A_r, \text{'?'})$ {Returning to parent: should no longer filter transactions with $A_r$}
21:  Store $Nd$ locally keyed by Node ID
22:  return Node ID of interior node $Nd$ {Execution continues at site owning parent node}
23: else
24:   for $i = 1 \ldots p$ do
25:     Randomly choose r from the range of random numbers for the cryptographic system
26:     $lv_i \leftarrow Encrypt(0, r)$ {Random encryption of 0}
27:   end for
28:   Create leaf node Nd with $p$ values $lv_1, \ldots, lv_p$
29:   return Node ID of leaf node $Nd$
30: end if
```

The selected attribute can be numerical or categorical. For numeric attributes, a split point π is randomly chosen within the range of the selected attribute values and the `BuildTree` is called twice: first, to generate a left subtree corresponding to the attribute values $< \pi$; second to generate a right subtree corresponding to the attribute values $>= \pi$ (lines 5-12 of Algorithm 4). For categorical attribute, `BuildTree` is called for each value of the attribute corresponding to the current node (lines 14-18 of Algorithm 4). This process continues until the level of the tree reaches the given depth (lines 23-29 of Algorithm 4). At this point a leaf node is created by the site owning the class attribute. Each of the class values is initialized to random encryption of zero using threshold-based additively homomorphic encryption [11]. Additively homomorphic encryption is used to enable computation of the sum of encrypted values for updating/incrementing the class statistics in Algorithm 5. Also, random encryption prevents disclosure of which class value is updated/incremented during statistics updating phase.

5.1.1 Update Statistics

For updating the statistics, in the original RDT algorithm, for each random tree and instance in the training data set, the tree is traversed to the appropriate leaf node. At the leaf node, the element in the class distribution vector corresponding to the class label of the instance is incremented by one. This process is repeated for all the random trees.

```
Algorithm 5 UpdateStatistics()
Require: Transaction set $T$ partitioned vertically between sites $P_1, \dots, P_k$
Require: $P_u$ holds $n_u$ attributes
Require: $p$ class values, $c_1, \dots, c_p$, with $P_k$ holding the class attribute
Require: $m$, the number of random trees built (let $nodeid_j$ represent the root node of the $j$th tree)
1: for each instance $t_x$ do
2:   {At $P_k$, build the class vector}
3:   for $i = 1 \dots p$ do
4:     Randomly choose $r$ from the set of positive integers
5:     if $t_x$ has class $c_i$ then
6:       $lv_i \leftarrow Encrypt(1, r)$ {Random encryption of 1}
7:     else
8:       $lv_i \leftarrow Encrypt(0, r)$ {Random encryption of 0}
9:     end if
10:  end for
11:  for $j = 1 \dots m$ do
12:    $incrementStats(t_x, nodeId_j, lv)$
13:  end for
14: end for
```

However, in the vertically partitioned data, the nodes and the attributes of the Transaction set are distributed across multiple sites. Moreover, while updating statistics no site should learn the attributes and attribute values in the data set of other sites. The algorithm for update statistics for vertically partitioned data is given in Algorithm 5 that calls `incrementStats` for distributed tree traversal and incrementing the class statistics using additively homomorphic encryption for preventing information leakage. In the update statistics algorithm (Algorithm 5, first the site that owns the class attribute builds a class vector corresponding to the given instance (lines 2-10 of Algorithm 5). In the class vector, the element corresponding to the class label of the instance stores a random encryption of '1'; while all other elements store random encryption of '0'. With random encryption, all the elements in the class vector will have distinct values to prevent other sites to learn about the class label of the given instance during tree traversal. These random encryptions are created through additively homomorphic public key encryption, such as Paillier encryption [12].

After building the class vector for the given instance, each of the distributed random trees is traversed by passing control from site to site based on the decision made. This tree traversal (lines 7-20 of Algorithm 6) starts from the site that has the root node of the tree. For the given instance, each site knows the attribute values for the nodes at its site and can thus evaluate the branch. Once the leaf node is reached, the class statistics of the leaf node are updated by multiplying each of the element in the class distribution vector of the leaf node by the corresponding element in the class vector generated for the given instance (lines 2-6 of Algorithm 6). As all the elements in the class distribution vector of the leaf node and in the class vector of the given instance are encrypted using homomorphic encryption key, this element-wise multiplication results in the element-wise addition of the plain text values of the two vectors. This follows from the following fundamental property of additively homomorphic encryption: Given a homomorphic encryption key E_{pk} and two plain text messages m_1 and m_2, $E_{pk}(m_1 + m_2) := E_{pk}(m_1)E_{pk}(m_2)$ As a result of this element-wise addition, the plain text value of the element in the class distribution vector corresponding to the class label of the instance is incremented by one.

```
Algorithm 6 incrementStats(instId, nodeId, lv): increments 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:   {update leaf values in $nodeId$}
4:   for $i = 1 \dots p$ do
5:     $nodeId.val_i \leftarrow nodeId.val_i * lv_i$ {Increment by 1 or hide}
6:   end for
7: else
8:   {$nodeId$ is an interior node}
9:   $Nd \leftarrow$ local node with id $nodeId$
10:  if Nd.A is numeric then
11:    if value of attribute $Nd.A$ for transaction $instId$ is smaller than split point then
12:      $childId \leftarrow Nd.leftsubtree$
13:    else
14:      $childId \leftarrow Nd.rightsubtree$
15:    end if
16:  else
17:    $value \leftarrow$ the value of attribute $Nd.A$ for transaction $instId$
18:    $childId \leftarrow Nd.value$
19:  end if
20:  childId.Site.$incrementStats(instId, childId, lv)$ {Actually tail recursion: this site need never learn the class}
21: end if
```

```
Algorithm 7 classifyInstance(instId)
Require: $m$, the number of random trees built (let $nodeid_j$ represent the root node of the $j$th tree)
1: for $i = 1 \dots p$ do
2:   Randomly choose $r$ from the range of random numbers for the cryptographic system
3:   $lv_i \leftarrow Encrypt(0, r)$ {Random encryption of 0}
4: end for
5: for $j = 1 \dots m$ do
6:   $currstats \leftarrow retriveStats(instId, nodeId_j)$
7:   for $i = 1 \dots p$ do
8:     $lv_i \leftarrow lv_i * currstats_i$
9:   end for
10: end for
11: Collaboratively decrypt each $lv_i$ and divide by m to get actual statistics
```

5.1.2 *Performing Classification*

Instance classification also proceeds in a distributed fashion similar to update statistics. For instance classification, the prediction of the given instance is computed by taking an average of the probability outputs from multiple RDTs. The distributed algorithm for instance classification is given in Algorithm 7.

We explain the instance classification procedure using an illustrative example with the two RDTs of depth 2 shown in

Fig. 3. Assume today is sunny with high humidity, high temperature, and weak wind. The classification would proceed as follows. The root node of the first tree is with site `S1` and that of the second tree is with `S2`. For the first tree, Site `S1` retrieves the attribute for `S1L1:Outlook`. Since `S1` knows the forecast is sunny, the token `S2L2` is retrieved. This token is with site `S2` and so the control passes to `S2` which retrieves the attribute for `S2L2: Humidity`. The humidity forecast is high, so the branch to the leaf node `S2L3` is taken that stores the encrypted class distribution probabilities: $E(0)$ for "yes" and $E(3)$ for "no." Similarly, instance classification with the second random tree leads to the leaf node `S2L6` with class distribution probabilities: $E(1)$ for "yes" and $E(2)$. These can now be homomorphically added together and averaged to give the predicted class distribution.

Algorithm 8 retrieveStats(instId, nodeId): returns the class/distribution for the instance represented by instId in tree given by nodeId

```
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:      if Nd.A is numeric then
7:          if value of attribute $Nd.A$ for transaction $instId$
            is smaller than split point then
8:              $childId \leftarrow Nd.leftsubtree$
9:          else
10:             $childId \leftarrow Nd.rightsubtree$
11:         end if
12:     else
13:         $value \leftarrow$ the value of attribute $Nd.A$ for transaction $instId$
14:         $childId \leftarrow Nd.value$
15:     end if
16:     return $childId.Site.retrieveStats(instId, childId)$
        {Actually tail recursion: this site need never learn
        the class}
17: end if
```

6 Analysis

We first derive the computation and communication cost, assuming k sites, n attributes, j instances, p class values, and m random trees, and then go through the security analysis.

6.1 Computation Cost

Since we are particularly interested in comparing against cryptographic algorithms, we only count the number of cryptographic operations (encryption, decryption, exponentiation). In general, the non-cryptographic operations do not incur much computation overhead as compared to the cryptographic operations, so their overhead can be ignored.

For horizontally partitioned data, only the leaf nodes are encrypted, by each party. Since the depth of a tree is $n/2$ (assuming binary splits), there are $2^{n/2-1}$ leaf nodes. With p classes, m trees, and k parties, there are a total of $O(pmk2^{n/2-1})$ encryptions. Classifying a new instance requires $O(mp)$ homomorphic multiplications and p threshold decryptions. For vertically partitioned data, all leaf nodes are originally encrypted and then updated. Thus, there are $O(mp2^{n/2-1})$ encryptions and $O(jp)$ homomorphic multiplications. Classifying a new instance requires p threshold decryptions.

6.2 Communication Cost

For horizontally partitioned data, the encrypted leaf values for the entire trees can be exchanged in one message. Thus, for the tree building phase $O(mk)$ messages are exchanged. For classifying a new instance $O(k)$ messages are exchanged for the threshold decryption. In the case of vertically partitioned data, for building a tree, in the worst case, a message is exchanged for every node. Since there are n attributes, the depth of each tree is $n/2$, thus giving us $O(2^{n/2})$ nodes (assuming binary splits). Therefore, in general, $O(m2^{n/2})$ messages for the tree building phase. For updating the statistics, each instance travels down to the corresponding leaf node. Thus, we have $n/2$ messages for each instance for each tree, giving $O(nmj)$ messages. Similarly, classifying an instance requires $O(nm)$ messages.

6.3 Security Analysis

We first look at the horizontal case. In case (1c), the tree structure is known to everyone, though the leaf class distribution vector for all trees are encrypted using the threshold encryption. Since semantically secure homomorphic encryption is used, this reveals no information to any of the parties about the values. Furthermore, since threshold encryption is used, none of the encrypted values can be decrypted without interaction from the minimum (threshold) number of parties. The threshold is a parameter that can be set (i.e., for (t, k) threshold encryption, it is necessary to have interaction between at least t of the k parties). If the threshold is set to k (the total number of parties), then none of the values can be decrypted even if all of the other parties collude against one party. However, when an instance is classified, the class distribution vector corresponding to the sum is computed and revealed to the party. It is important to note that this does not reveal more information than revealing the average (since the sum can be computed from the average), though it does reveal more information than a normalized class probability distribution (namely, how many other instances in the training data reach the corresponding leaf node). Every classification essentially reveals one linear equation in the global class distribution vector for the m leafs. Given enough linear equations, it may be possible to solve the system of linear equations to find the actual class distribution vectors corresponding to the leaf node. However, this will still be only the global values, and it is impossible to accurately estimate the *local* values (since infinite solutions are possible), unless every party colludes against one party.

Note that in such a case the information leaked is the number of records of a particular class reaching a particular leaf node. It is possible to protect this information from leaking. For example, one possibility is to multiply the encrypted vector by a positive (unknown) random number to hide the information about the number of participating records. This can be done by having each party multiply the global encrypted vector by a random amount. This will still preserve the distribution of class values. Indeed, in this sense model privacy is not preserved since the proportion of classes will be observable to the adversary. However, note that this is unavoidable if the class distribution is to be reported (is part of the output). If the classifier were

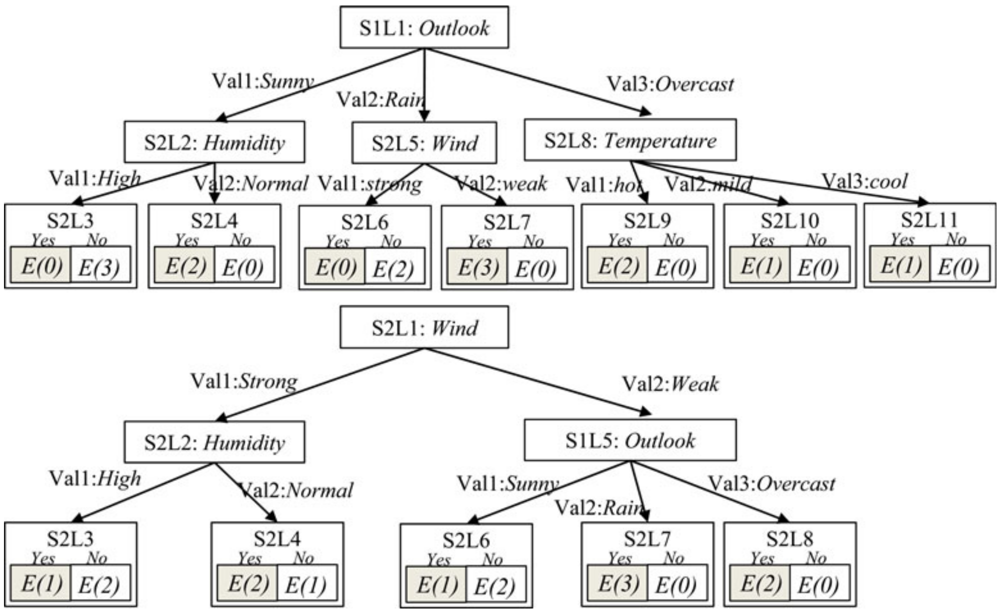

Fig. 3. The random trees built on vertical partitioning.

supposed to only report the predicted class instead of the class distribution, this information leakage could be prevented (by carrying out comparisons over the encrypted values, and only reporting the class with the largest value).

In the case of vertical partitioning, even the tree structure is unknown to any of the parties. The process of building a tree does not leak any information except what is leaked through classifying an instance (because this is the basic underlying operation). However, each classification does reveal some limited information about the branch taken to reach the leaf (since the branch must be limited to the values contained within the instance). Even in this case, since each new instance is also distributed, it is impossible to completely reconstruct the entire tree without the help of all parties.

7 Experimental Evaluation

The experiments were performed in a distributed environment wherein each node was a Linux box with 4 GB memory and 2.8 GHz Intel Xeon processor. The nodes were connected in a star topology over 1 Gbit Ethernet.

7.1 Adapting Weka

Weka [13] developed at the University of Waikato in New Zealand, is a collection of machine learning algorithms for data mining tasks implemented in Java. Apart from providing algorithms, it is a general implementation framework, along with support classes and documentation. It is extensible and convenient for prototyping purposes. However, the Weka system is a centralized system meant to be used at a single site.

We have implemented our Privacy Preserving RDT algorithm and integrated it into Weka version 3.6. We follow the general model of operation to extend Weka for privacy preserving distributed classification of Vaidya et al. [7]. Fig. 4 demonstrates the basic interaction diagram. Java RMI is used to implement the distributed protocol between all of the sites.

7.2 Experimental Results

Experiments were carried out on four data sets from the UCI Machine Learning Repository: *Mushroom*, *Nursery*, *Image Segmentation*, and *Car*. The experimental results for both horizontally partitioned and vertically partitioned data are presented in Table 2. Specifically, the results compare RDT model building time (BT), instance classification time (CT), and classification accuracy (Acc) by varying the number of sites (3, 4, 5) as well as by varying number of trees (7, 10, 14, 20) for *Mushroom*, *Nursery* and *Image Segmentation*. Tables 2a and 2b show the results for the horizontally and vertically partitioned case respectively. In each data set, 90 percent of the instances were used for training and 10 percent for testing. When split between the parties horizontally, we assume that each party has an equal number of instances. Similarly, when split vertically, we assume that each party has an approximately equal number of attributes. *Nursery* has eight attributes (all categorical) and 12,960 instances; *Mushroom* has 22 attributes (all categorical) and 8,124 instances; *Image Segmentation* has 19 attributes (all numeric) and 2,310 instances; *Car* has six attributes (all categorical) and 1,728 instances; For *Nursery* the depth of all the RDTs was 4 for both horizontal partitioned and vertical partitioned case. For *Mushroom* the depth of RDTs was 8 for the horizontal partitioned case and reduced to 4 for the vertical partitioned case (to ensure sufficient memory for tree construction). We also grouped the category values of the attributes for *Mushroom* so that the domain cardinality of the attributes was no more than 4. For *Image Segmentation* the depth of RDTs was 10 for both horizontal partitioned and vertical partitioned case. All experiments were repeated three times and results were averaged.

Table 2 shows that the time taken to build the classifier scales linearly with the number of trees, as does the time taken to classify a new instance. This is true regardless of

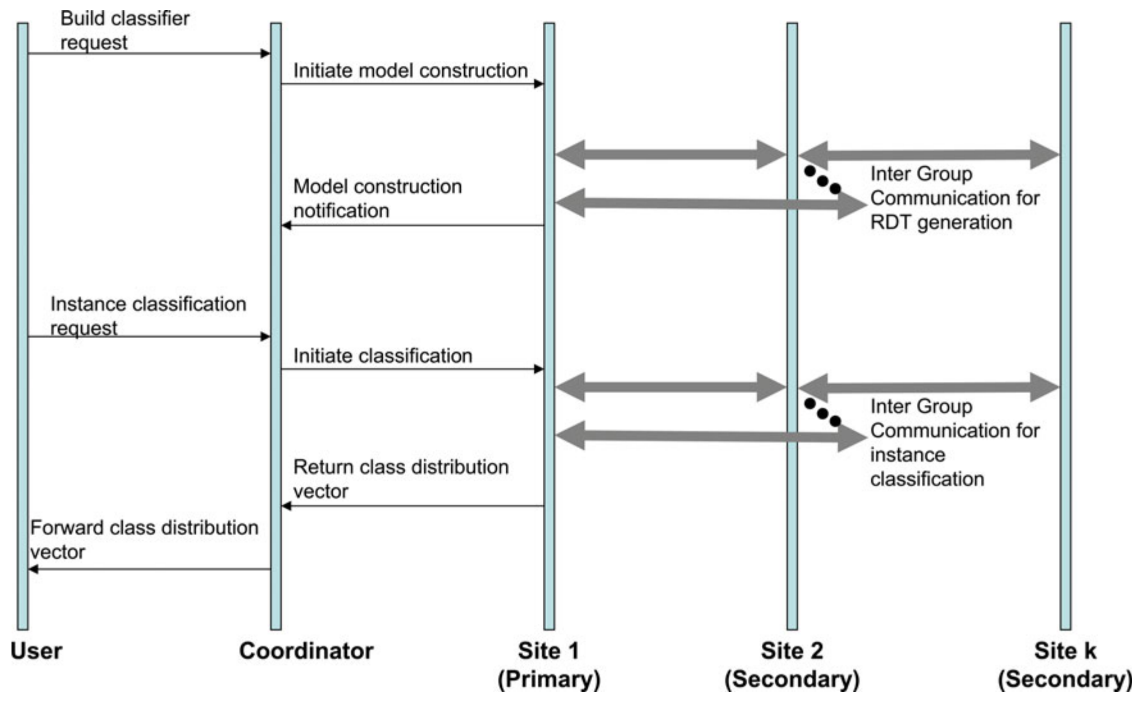
Build classifier request
Initiate model construction
Inter Group Communication for RDT generation
Model construction notification
Instance classification request
Initiate classification
Inter Group Communication for instance classification
Return class distribution vector
Forward class distribution vector
User
Coordinator
Site 1 (Primary)
Site 2 (Secondary)
Site k (Secondary)

Fig. 4. Basic interaction diagram.

whether the underlying attributes are numeric or categorical. Note though, that the time required for vertically partitioned data (in both cases) is orders of magnitude higher than that required for horizontal partitioned data. This is since the time reported includes the time for computation and for communication, and is to be expected since the actual tree is distributed in the vertical partitioned case. Indeed, the degree of communication required for vertically partitioned data is proportional to the number of instances, where as it only depends on the number of parties for horizontally partitioned data. Thus, most of the time taken is due to the additional Java RMI invocations required for the vertical partitioned case. Note that once the relatively expensive protocol to build the RDT classifier has already been executed, it is a computationally trivial task to classify any given instance. Also, the classification accuracy increases as the number of trees increase from 7-20.

For *Nursery* and *Image Segmentation*, the accuracy results are similar for both horizontally and vertically partitioned cases since the depth of the RDTs was the same in both cases. The small variation is due to the random choice of testing instances. For *Mushroom*, the classification accuracy in the horizontal partitioned case (RDT depth 8) is significantly greater than vertical partitioning (RDT depth 4). This is due to the difference in RDT depth.

Table 2 also shows that the time taken for training and classification in vertically partitioned data both increase as the number of sites increases. This is due to the additional communication (and RMI calls) required. The accuracy stays the same. Table 3 compares the time taken by ID3 and RDT for all data sets assuming vertical partitioning among three sites. The secure version of ID3[7] was used in the same distributed environment. From the results, it can be noted that building an RDT model is generally an order of magnitude faster than building the ID3 model. However, instance classification is significantly slower. This is because only one tree is used for classifying an instance using the ID3 model; whereas, multiple RDTs (10 or more) are used for classification. Nevertheless, the classification speed is reasonable. Note that the accuracy of RDT is worse in this case than ID3. However, the accuracy increases with larger number of trees and in general, is comparable to other inductive learning models [2], [3], [4].

7.3 Experimental Conclusion

The distributed RDT algorithms and implementation presented in this paper are a significant step forward in creating usable, distributed, privacy-preserving, data mining algorithms. The running time of the algorithms, is comparatively much faster than the existing implementations, and is usable on everyday computing hardware. As compared to the standard, non privacy-preserving version, the accuracy of the privacy-preserving solution is exactly the same, though the computational overhead is significant. However, privacy is not free. In general, privacy-preserving protocols are more expensive than non-privacy-preserving protocols for the same problem. For example, [7] shows that the privacy-preserving ID3 requires two orders of magnitude larger computation time than the non privacy-preserving version. Indeed, this motivated us to build the more efficient solutions proposed in this paper, so that use of PPDM solutions can become a reality.

8 Related Work

There has been extensive work in privacy-preserving data mining. The perturbation approach, pioneered by Agrawal and Srikant [14], relies on adding "noise" to the data before the data mining process. Techniques are then used to somehow remove the noise from the data mining results. Several

TABLE 2
Experimental Results for the *Mushroom, Nursery, Image Segmentation* Comparing RDT Model Building Time (BT), Instance Classification Time (CT), and Classification Accuracy (Acc)

(a) Horizontally partitioned data

Nursery (No. of attributes = 8; No. of instances = 12,960; RDT Depth = 4)									
	3 sites			**4 sites**			**5 sites**		
	BT seconds	**CT** seconds	**Acc** %	**BT** seconds	**CT** seconds	**Acc** %	**BT** seconds	**CT** seconds	**Acc** %
7 trees	29.5	0.044	87.1	27.8	0.049	87.2	23.9	0.055	87.7
10 trees	33.8	0.046	86.6	29.4	0.049	87.3	27.2	0.055	90.8
14 trees	33.4	0.045	89.3	30.3	0.050	89.5	28.1	0.054	88.8
20 trees	37.9	0.046	89.6	33.5	0.051	88.0	32.3	0.056	92.2
Mushroom (No. of attributes = 22; No. of instances = 8124; RDT depth = 8)									
	3 sites			**4 sites**			**5 sites**		
7 trees	125.5	0.032	96.7	112.5	0.036	96.7	146.5	0.041	97.3
10 trees	177.3	0.033	97.6	190.2	0.038	96.9	196.7	0.044	98.2
14 trees	236.9	0.033	98.4	225.9	0.038	97.2	259.8	0.043	99.3
20 trees	349.2	0.035	99.0	379.4	0.044	98.5	342.4	0.045	99.1
Image Segmentation (No. of attributes = 19; No. of instances = 2310; RDT depth = 10)									
	3 sites			**4 sites**			**5 sites**		
7 trees	46.8	0.022	70.5	42.7	0.025	70.3	50.5	0.027	70.9
10 trees	62.2	0.024	74.8	64.4	0.026	75.0	66.5	0.028	75.7
14 trees	87.3	0.024	77.1	82.9	0.026	77.7	83.5	0.029	77.9
20 trees	110.5	0.025	81.4	110.2	0.027	80.9	105.7	0.031	81.3

(b) Vertically partitioned data

Nursery (No. of attributes = 8; No. of instances = 12,960; RDT Depth = 4)									
	3 sites			**4 sites**			**5 sites**		
	BT seconds	**CT** seconds	**Acc** %	**BT** seconds	**CT** seconds	**Acc** %	**BT** seconds	**CT** seconds	**Acc** %
7 trees	19,112	0.230	84.6	24,093	0.320	83.1	21,674	0.296	82.4
10 trees	24,489	0.353	89.1	30,817	0.387	90.1	33,350	0.417	84.0
14 trees	32,750	0.410	93.1	41,675	0.460	90.0	46,320	0.558	87.3
20 trees	52,113	0.626	89.3	56,968	0.653	90.2	66,158	0.758	85.2
Mushroom (No. of attributes = 22; No. of instances = 8124; RDT depth = 4)									
	3 sites			**4 sites**			**5 sites**		
7 trees	8,151	0.218	88.5	7,686	0.214	87.4	8,950	0.231	90.0
10 trees	10,481	0.257	93.1	11,707	0.283	88.6	12,487	0.295	93.6
14 trees	14,342	0.327	94.7	15,470	0.342	88.0	16,617	0.361	87.3
20 trees	19,798	0.426	89.0	22,965	0.457	90.1	25,143	0.498	87.7
Image Segmentation (No. of attributes = 19; No. of instances = 2310; RDT depth = 10)									
	3 sites			**4 sites**			**5 sites**		
7 trees	1410	0.192	70.5	1566	0.259	69.6	1575	0.301	70.6
10 trees	2077	0.222	75.7	2083	0.261	76.2	2250	0.328	76.8
14 trees	2552	0.233	77.5	2823	0.295	77.3	3207	0.364	78.7
20 trees	3849	0.283	81.9	4074	0.335	80.4	4851	0.441	81.2

privacy-preserving data mining algorithms have since been proposed [1], [15]. One point of concern is that, since the original data is still available (though in perturbed form), noise removal techniques can be used to give estimates of the original values, giving rise to debate about the security properties of such algorithms [16], [17].

The alternative approach to protecting privacy of distributed sources using cryptographic techniques was first applied in the area of data mining for the construction of decision trees by Lindell and Pinkas [18]. This work falls under the framework of secure multiparty computation [8], achieving "perfect" privacy, i.e., nothing is learned

TABLE 3
ID3 [7] and RDT Comparison

Car Dataset: No. of attributes = 6; No. of instances = 1728; No. of instances used for training = 1545; No. of instances used for testing = 183					
ID3			**RDT** (10 trees; depth 4)		
BT seconds	**CT** seconds	**Acc** %	**BT** seconds	**CT** seconds	**Acc** %
3041.26	0.0057	85.55	509.52	0.091	71.6
Nursery Dataset: No. of attributes = 8; No. of instances = 2000; No. of instances used for training = 1791; No. of instances used for testing = 209					
BT	**CT**	**Acc**	**BT**	**CT**	**Acc**
9084.1	0.0244	89	613.64	0.0968	83.2
Mushroom Dataset: No. of attributes = 22; No. of instances = 2060; No. of instances used for training = 1844; No. of instances used for testing = 216					
BT	**CT**	**Acc**	**BT**	**CT**	**Acc**
1600.24	0.0158	98.06	776.21	0.0932	89.4

that could not be deduced from one's own data and the resulting tree. The key insight was to trade off computation and communication cost for accuracy, improving efficiency over the generic secure multiparty computation method. However, the proposed solution is still too inefficient for practical usage.

There has been work in distributed construction of decision trees on vertically partitioned data. Wang et al. present a solution based on passing the transaction identifiers between sites [19]; while this does not reveal specific attribute values, parties learn which transactions follow which path down the tree, enabling (for example) one site to say "these two individuals have the same attribute values." Du and Zhan [20] do suggest a way of building a privacy-preserving decision tree classifier for vertically partitioned data. Their method is limited to two parties, assumes that both parties have the class attribute, and is not implemented. Vaidya et al. [7] extend this to the multi-party case, also relaxing the assumption that the class attribute must be known to all parties. This approach has been implemented, though the experimental results suggest that for large scale data, the computational complexity is quite high. Additional classification techniques include protocols to build a Naïve Bayes classifier [21], while Wright and Yang [22] propose a similar protocol for learning the Bayesian network structure. These works make trade-offs between efficiency and information disclosure, however all maintain provable bounds on disclosure. Jagannathan et al. [6] propose ways to construct a differentially private RDT classifier from a centralized data set. Since our data is distributed, we cannot directly use this. There has also been work to address association rules in horizontally partitioned data [23], [24], EM Clustering in horizontally partitioned data [25], K-means clustering in vertically partitioned data [26], [27], association rules in vertically partitioned data [28], [29], and generalized approaches to reducing the number of "on-line" parties [30].

The methodology for proving the correctness of the algorithm comes from secure multiparty computation [8], [31], [32]. Recently, there has been a renewed interest in this field, a good discussion can be found in [33]. Currently, assembling these into efficient privacy-preserving data mining algorithms, and proving them secure, is a challenging task. This paper demonstrates how we can leverage the beneficial properties of both randomization and cryptography to provide a highly efficient yet secure approach to performing classification. We hope that this approach will be pursued in future work to make other privacy-preserving data mining tasks more efficient.

9 Conclusion

We have demonstrated that general and efficient distributed privacy preserving knowledge discovery is truly feasible. We have considered the security and privacy implications when dealing with distributed data that is partitioned either horizontally or vertically across multiple sites, and the challenges of performing data mining tasks on such data. Since RDTs can be used to generate equivalent, accurate and sometimes better models with much smaller cost, we have proposed distributed privacy-preserving RDTs. Our approach leverages the fact that randomness in structure can provide strong privacy with less computation. The experiments show that the privacy preserving version of the RDT algorithm scales linearly with data set size, and requires significantly less time than alternative cryptographic approaches. In the future, we plan to develop general solutions that can work for arbitrarily partitioned data.

Acknowledgments

The work of Danish Mehmood and Basit Shafiq was supported in part by the LUMS Departmental Research Grant. The work of Jaideep Vaidya and David Lorenzi was supported in part by a Rutgers Business School—Newark and New Brunswick, Research Resources Committee Grant.

References

[1] J. Vaidya, C. Clifton, and M. Zhu, *Privacy-Preserving Data Mining*.ser. Advances in Information Security first ed., vol. 19, Springer-Verlag, 2005.

[2] W. Fan, H. Wang, P.S. Yu, and S. Ma, "Is Random Model Better? On Its Accuracy and Efficiency," *Proc. Third IEEE Int'l Conf. Data Mining (ICDM '03)*, pp. 51-58, 2003.

[3] W. Fan, J. McCloskey, and P. S. Yu, "A General Framework for Accurate and Fast Regression by Data Summarization in Random Decision Trees," *Proc. 12th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '06)*, pp. 136-146, 2006.

[4] X. Zhang, Q. Yuan, S. Zhao, W. Fan, W. Zheng, and Z. Wang, "Multi-Label Classification without the Multi-Label Cost," *Proc. SIAM Int'l Conf. Data Mining (SDM '10)*, pp. 778-789, 2010.

[5] A. Dhurandhar and A. Dobra, "Probabilistic Characterization of Random Decision Trees," *J. Machine Learning Research*, vol. 9, pp. 2321-2348, 2008.

[6] G. Jagannathan, K. Pillaipakkamnatt, and R.N. Wright, "A Practical Differentially Private Random Decision Tree Classifier," *Proc. IEEE Int'l Conf. Data Mining Workshops (ICDMW '09)*, pp. 114-121, 2009.

[7] J. Vaidya, C. Clifton, M. Kantarcioglu, and A.S. Patterson, "Privacy-Preserving Decision Trees over Vertically Partitioned Data," *ACM Trans. Knowledge Discovery from Data*, vol. 2, no. 3, pp. 1-27, 2008.

[8] O. Goldreich, "General Cryptographic Protocols," *The Foundations of Cryptography*, vol. 2, pp. 599-764, Cambridge Univ. Press, 2004.

[9] D. Gritzalis, *Secure Electronic Voting*.ser. Advances in Information Security first ed., vol. 7, Springer-Verlag, 2003.

[10] B. Schneier, *Applied Cryptography*. second ed., John Wiley & Sons, 1995.

[11] R. Cramer, I. Damgard, and J.B. Nielsen, "Multiparty Computation from Threshold Homomorphic Encryption," *Proc. Int'l Conf. Theory and Application of Cryptographic Techniques (EUROCRYPT '01)*, pp. 280-299, May 2001.

[12] P. Paillier, "Public Key Cryptosystems Based on Composite Degree Residuosity Classes," *Proc. 17th Int'l Conf. Theory and Application of Cryptographic Techniques (EUROCRYPT '99)*, pp. 223-238, 1999.

[13] I.H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, Oct. 1999.

[14] R. Agrawal and R. Srikant, "Privacy-Preserving Data Mining," *Proc. ACM SIGMOD Conf. Management of Data*, pp. 439-450, May 2000.

[15] D. Agrawal and C.C. Aggarwal, "On the Design and Quantification of Privacy Preserving Data Mining Algorithms," *Proc. 20th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems*, pp. 247-255, May 2001.

[16] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar, "On the Privacy Preserving Properties of Random Data Perturbation Techniques," *Proc. Third IEEE Int'l Conf. Data Mining (ICDM '03)*, Nov. 2003.

[17] Z. Huang, W. Du, and B. Chen, "Deriving Private Information from Randomized Data," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, June 2005.

[18] Y. Lindell and B. Pinkas, "Privacy Preserving Data Mining," *J. Cryptology*, vol. 15, no. 3, pp. 177-206, 2002.

[19] K. Wang, Y. Xu, R. She, and P.S. Yu, "Classification Spanning Private Databases," *Proc. 21st Nat'l Conf. Artificial Intelligence*, pp. 293-298, 2006.

[20] W. Du and Z. Zhan, "Building Decision Tree Classifier on Private Data," *Proc. IEEE Int'l Conf. Data Mining Workshop on Privacy, Security, and Data Mining*, pp. 1-8, Dec. 2002.

[21] J. Vaidya, M. Kantarcioglu, and C. Clifton, "Privacy Preserving Naive Bayes Classification," *Int'l J. Very Large Data Bases*, vol. 17, no. 4, pp. 879-898, July 2008.

[22] R. Wright and Z. Yang, "Privacy-Preserving Bayesian Network Structure Computation on Distributed Heterogeneous," *Proc. 10th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, Aug. 2004.

[23] M. Kantarcioĝlu and C. Clifton, "Privacy-Preserving Distributed Mining of Association Rules on Horizontally Partitioned Data," *IEEE Trans. Knowledge and Data Eng.*, vol. 16, no. 9, pp. 1026-1037, Sept. 2004.

[24] M. Kantarcioglu and C. Clifton, "Privacy-Preserving Distributed Mining of Association Rules on Horizontally Partitioned Data," *Proc. ACM SIGMOD Workshop Research Issues on Data Mining and Knowledge Discovery (DMKD '02)*, pp. 24-31, June 2002.

[25] X. Lin, C. Clifton, and M. Zhu, "Privacy Preserving Clustering with Distributed EM Mixture Modeling," *J. Knowledge and Information Systems*, vol. 8, no. 1, pp. 68-81, July 2005.

[26] J. Vaidya and C. Clifton, "Privacy-Preserving k-Means Clustering over Vertically Partitioned Data," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, pp. 206-215, Aug. 2003.

[27] G. Jagannathan and R.N. Wright, "Privacy-Preserving Distributed k-Means Clustering over Arbitrarily Partitioned Data," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, pp. 593-599, Aug. 2005.

[28] J. Vaidya and C. Clifton, "Privacy Preserving Association Rule Mining in Vertically Partitioned Data," *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, pp. 639-644, July 2002.

[29] J. Vaidya and C. Clifton, "Secure Set Intersection Cardinality with Application to Association Rule Mining," *J. Computer Security*, vol. 13, no. 4, pp. 593-622, Nov. 2005.

[30] M. Kantarcioglu and J. Vaidya, "An Architecture for Privacy-Preserving Mining of Client Information," *Proc. IEEE Int'l Conf. Data Mining Workshop on Privacy, Security, and Data Mining*, pp. 37-42, Dec. 2002.

[31] A.C. Yao, "How to Generate and Exchange Secrets," *Proc. 27th IEEE Symp. Foundations of Computer Science*, pp. 162-167, 1986.

[32] O. Goldreich, S. Micali, and A. Wigderson, "How to Play Any Mental Game—A Completeness Theorem for Protocols with Honest Majority," *Proc. 19th ACM Symp. Theory Computing*, pp. 218-229, 1987.

[33] W. Du and M.J. Atallah, "Secure Multi-Party Computation Problems and Their Applications: A Review and Open Problems," *Proc. New Security Paradigms Workshop (NSPW '01)*, pp. 11-20, Sept. 2001.

Jaideep Vaidya received the BE degree in computer engineering from the University of Mumbai, India, and the MS and PhD degrees in computer science from Purdue University. He is currently an associate professor in the MSIS Department, Rutgers University. His general area of research is in data mining, data management, security, and privacy. He has published more than 90 technical papers in peer-reviewed journals and conference proceedings, and has received three best paper awards from the premier conferences in data mining and databases. He is also the recipient of a US National Science Foundation (NSF) Career Award and is a member of the ACM and a senior member of the IEEE.

Basit Shafiq received the BS degree in electronic engineering from the GIK Institute of Engineering Sciences and Technology, Pakistan, the MS and PhD degrees in electrical and computer engineering from Purdue University. He is currently an assistant professor in the Computer Science Department, Lahore University of Management Sciences (LUMS). Prior to joining LUMS, he was a Research Assistant Professor at the Center for Information Management, Integration and Connectivity (CIMIC), Rutgers University. His research interests include information systems security and privacy, access-control management in distributed systems, Web services composition and verification, and ontologies. He is a member of the IEEE.

Wei Fan received the PhD degree in computer science from Columbia University in 2001 and had been working in IBM T.J. Watson Research since 2000. He is an associate director of Huawei Noah's Ark Lab. His main research interests and experiences include various areas of data mining and database systems, such as, stream computing, high performance computing, risk analysis, ensemble methods, graph mining, transfer learning, bioinformatics, and social network analysis. He is a member of the IEEE.

Danish Mehmood received the bachelor's degree in computer science from the Lahore University of Management Sciences in 2011. He is currently working toward the master's degree at the David R. Cheriton School of Computer Science, University of Waterloo. His research interests include Privacy Enhancing Technologies.

David Lorenzi received the BS degree in computer information Systems from Arizona State University. He is currently a doctoral student in information technology at Rutgers University. His research interests include privacy, security, and digital government.