Privacy Preserving Client/Vertical-Servers Classification

Derian Boer, Zahra Ahmadi(✉), and Stefan Kramer

Institut für Informatik, Johannes Gutenberg-Universität,
Staudingerweg 9, 55128 Mainz, Germany
dboer@students.uni-mainz.de, {zaahmadi,kramer}@informatik.uni-mainz.de

Abstract. We present a novel client/vertical-servers architecture for hybrid multi-party classification problem. The model consists of clients whose attributes are distributed on multiple servers and remain secret during training and testing. Our solution builds privacy-preserving random forests and completes them with a special private set intersection protocol that provides a central commodity server with anonymous conditional statistics. Subsequently, the private set intersection protocol can be used to privately classify the queries of new clients using the commodity server's statistics. The proviso is that the commodity server must not collude with other parties. In cases where this restriction is acceptable, it allows an effective method without computationally expensive public key operations, while it is still secure and avoids precision losses. We report the runtime results on some real-world datasets, and discuss different security aspects and finally give an outlook on further improvements.

Keywords: Vertically partitioned data · Private evaluation · Secure multi-party computation · Privacy preserving data mining · Random forest

1 Introduction

In the era of big data, the costs of storing and processing data is decreasing and as a result, the amount of collected data for analyzing purposes is also increasing. In this context, the goal is to make use of the potential knowledge, which additionally measured or mined data can provide. At the same time, the challenges of preserving privacy of personal and other sensitive data grow. This challenge becomes more important especially when partners collaborate with the intention to benefit from the union of their data. These partners can also be competitors, for example, and more or less trusted. Legal privacy concerns have to be considered as constraints in these cases. Aspects of privacy-preserving data include randomization, k-anonymity and l-diversity, downgrading application effectiveness, and secure multi-party computing (SMC) [1]. While randomization, k-anonymity and effectiveness downgrading require a trade-off between effectiveness (quality of the output) and privacy, SMC techniques do not effect

C. Alzate et al. (Eds.): MIDAS 2018/PAP 2018, LNAI 11054, pp. 125–140, 2019.
https://doi.org/10.1007/978-3-030-13463-1_10

the effectiveness. In SMC, the data sets remain completely private and hence need not be modified. Instead, special cryptographic communication protocols allow two or more parties to obtain aggregated results from their combined data, but each of them does not learn any information more than what can be derived from their common output. Thereby, the SMC algorithms lead to the same results as non-private algorithms do. This research area is also known as distributed privacy preservation, because the integrated data are partitioned on multiple parties who protect their shares. A special and upcoming case of secure multi-party computing is private evaluation, where a server has a sensitive model and a client has sensitive attribute vectors as input. The goal is that the client obtains a classification of her attribute vector with the use of the server's model, while the client does not see the model in plain text and the server is not able to get any parts of the client's input and output.

In this work, we consider a special case of the aforementioned private evaluation scenario for decision tree-based classification and set intersection in the secure multi-party computation scenario. We assume that the data of some clients is vertically partitioned and distributed across multiple servers. It is sufficient if at least one server knows the class values of the training instances. Each party can only see her attributes of the common decision trees. The leaf node statistics are stored by a trusted third party, which does not know any instances or tree attributes. To new test instances are classified with the following steps:

- The attribute vector of the client is vertically partitioned on the servers.
- The servers run a novel private set intersection protocol so that the client gets a shared sum of her leaf node identifier from each party. No server learns other attributes or information about the output at this step.
- Then a client can anonymously ask the commodity server about the class value statistics of her leaf node.

Our proposed architecture benefits from the following features:

1. A fast computation run time by removing computationally expensive primitives that are used in public key encryption methods such as Vaidya and Clifton's method [26],
2. Linear scaling with respect to the number of parties,
3. Accurate results in contrast to randomization based techniques [12,14],
4. Similar to private evaluation methods, no other party is able to learn any node statistics and cannot detect the similarities among different records, however, in contrast to them, our framework can handle distributed decision trees.

Moreover, there are several applications for this framework in spam filtering, crime reduction or credit assessment. Police forces, tax authority and financial institutions might be willing to cooperate in terms of fraud prevention, but only want to share uncoded personal data in case of reasonable suspicion. Another typical application is clinical diagnosis. In a real-world setting, the sensitive data of several institutions might be necessary to come up with a good diagnosis

for a client or a responsible expert, while none of them want to disclose their information to the others.

The rest of the paper establishes the proposed secure architecture in details. First, we give an overview on the background in the next section. Section 3 presents the problem setting and the proposed client/vertical server random forest model. We elaborate experimental results in Sect. 4. Finally, Sect. 5 concludes the paper.

2 Background

In the multi-party computing scenario, data can be partitioned among parties vertically, where the parties have different attributes of the same data objects, or horizontally, where the parties have different data objects of a compatible structure. There have been extensive studies on both partitioning approaches for privacy preserving decision tree based methods. In the rest of this section, we first provide an overview of decision tree induction and random forests, then, we briefly review the private decision tree learning literature for both vertically distributed data and private evaluation. As the focus of our work lies only on vertically partitioned data and private evaluation, we discard reviewing the horizontal private decision tree approaches [11,18,20].

ID3 and Random Forest: Decision trees are commonly used not only for solving classification and regression problems, but also for clustering with cluster descriptions [2]. A widely-used, intuitive decision tree algorithm is the ID3 algorithm [21] and its extension C4.5. Both use the Shannon entropy and information gain to create tree branches efficiently. The entropy can be replaced by other impurity measures with different sensitivities to costs [9]. Decision trees have shown promising results on many problems, nevertheless, their performance can be improved by a majority vote that combines the outcomes from many largely independent decision trees. A single tree is inclined to overfitting which causes a high variance. The random forest algorithm [5] tackles this instability by building several trees based on two randomization concepts: First, the training sets are varied by bootstrap aggregating, an equally-distributed random selection of records with replacement. Second, at each splitting step, it selects only s random attributes. This has the additional advantage that the trees can be constructed with very few or even without any data queries in the first step, if s is set to one. As a successful and well-interpretable learning model, in this paper, we focus on secure multi-party computation models for decision trees.

Secure Multi-party ID3 on Vertically Partitioned Data: The first work on vertically partitioned data for two parties was proposed by Du and Zahn [10]. In order to count the records that support a particular attribute or class value, they suggest that every party fills a binary vector with a one, if a record conforms with the currently examined attributes, and a zero, if a record does not. A secure shared two-party scalar product protocol and a secure shared logarithm protocol on these vectors allow to calculate the conditional entropy without revealing

the involved records. Similar to our architecture, these protocols are lightweight solutions, and require a commodity server that should not collude with any of the parties. However, this approach is hardly extensible to an n-party solution. The other drawback is the possibility of revealing sensitive data by making inferences from the public decision tree.

To solve the inference problem, some approaches use trees whose nodes are mapped to the attributes that are only visible for the corresponding party [23,26]. Vaidya and Clifton propose a private set intersection protocol (PSIP) [26], which can be applied in the same way as the scalar product protocol [10]. Their PSIP is based on public homomorphic encryption [8], and it requires a high number of key bits which increases the runtime significantly. Recently, some private set intersection protocols that use symmetric key encryption have been developed [16,17], however, the communication costs still remain relatively high. In another approach [23], each party finds the attributes with the highest information gain independently from the other parties at each branch step. The party with the attribute of the highest gain executes the split and broadcasts the separation of the records, but not the identity of the split attribute. This approach is only feasible if each party holds the class attribute. The other downside of this approach is that the similarities among different records can still be leaked. Both of these two approaches can support two or several parties.

Decision Trees on Randomized Data: Randomization approaches usually lead to faster results, but imply a trade-off between the individual's privacy and the quality of the results. The first randomization based multi-party tree induction used a multi-group randomized response (MRR) scheme [27] that works as follows: The attributes are partitioned into groups. In a first step, coin-flipping is conducted for each group and a user either tells the truth about all attributes in the same group or tells a lie about all of them. The trade-off between privacy and performance is regulated by a fixed probability of lie. One party works as a data collector of all randomized data sets and executes the ID3 algorithm on the collection. The cost of tree building and the accuracy loss can be reduced by employing a hybrid of MRR and SMC [25]. The ω attributes that have the highest information gains on the combined randomized data are selected and evaluated on a private dot product or intersection protocol.

Recently, many ϵ-differential solutions for ID3 [12,15] and tree ensembles [3, 14,19,22] were proposed. A solution is ϵ-differential private if the outcome of a calculation is insensitive to any particular record in the data set:

Definition 1. *A randomized computation M provides ϵ-differential privacy, if for any datasets A and B with symmetric difference $A \triangle B = 1$, and any set of possible outcomes $S \subseteq Range(M)$, $Pr[M(A) \in S] \leq Pr[M(B) \in S] \times e^{\epsilon}$.*

The randomization is obtained by the addition of noise, and the ϵ-parameter can be seen as a "privacy budget". As it fulfills the composability property, the parameters of consecutive queries can be accumulated. The major drawback, especially of the ID3 solutions, is the high variance in the accuracies, and that

only the individual's privacy is preserved, but not the private attribute distributions. Therefore, random decision trees have been shown to be more efficient and provide better security than ID3 induced trees in the context of differential privacy [22].

Private Tree Evaluation with a Client-Server Architecture: The field of privacy-preserving decision tree evaluation is a different, yet somewhat related to what we already discussed. Here, a server has a sensitive model and a client has sensitive attribute vectors as input. The goal is to classify the client's data while the sensitive inputs (model and query) remain hidden from the counterparty. Many approaches use homomorphic encryption [4,6,13,24]. Wu *et al.* [4] and Tai *et al.* [24] reduce the protocol complexity to be linear with respect to the number of decision nodes by representing the decision trees, which are high-degree polynomials, as linear functions.

3 Client/Vertical-Servers Random Forest

In this section, we explain our proposed client-server architecture in detail. First, we establish the problem definition. Then, we explain the private set intersection protocol, and how to build and apply the decision tree in a client-server architecture. Finally, we analyze the security of our proposed architecture.

3.1 Problem Setting

Problem Description: In this work, we consider the classification problem in the hybrid context of vertically partitioned data and the client-server architecture. Our proposed architecture is composed of two modes: a training phase in which the decision tree is built, and an evaluation mode where a test instance of a client is classified by the existing private model. In the training phase, we have m input vectors (training instances), whose attributes are distributed on n vertical servers p_i. Let $X_i[j]$ denote the attribute values of the record j that is known by p_i. The target value is held by the class server p_c, which can be one of the vertical servers, individual clients, or any other party. In the evaluation phase, one client c's query is classified by a trained model $f(x)$.

Constraints and Assumptions: The classification is a private service, such that no vertical server p_i is able to reveal any information about the attributes ($X_i[j]$) and the target value of c or similarities to other clients or training records. The client should not learn anything about the underlying model or any $X_i[j]$ than what can be deduced from $f(c)$. We allow the use of a semi-honest commodity server cs, which must not collude with any p_i and receives nothing but anonymous data.

Algorithm 1. Client/Vertical-Servers Set Intersection (CVSSI)

input : A vector $D_i \in \{0,1\}^s$ for each server p_i, $i \in \{1,...,n\}$, a collusion threshold parameter T

output: A vector $Y \in \{0,1\}^s$, where $Y[j] = \prod_{i=1}^{n} D_i[j]$

1 **for** $j \leftarrow 1$ **to** s **do**
2 **foreach** server p_i **do**
3 $r_i^0[j] \leftarrow$ new random number
4 **for** $t \leftarrow 1$ **to** T **do**
5 $r_i^t[j] \leftarrow$ new random number
6 send $r_i^t[j]$ to server $p_{i+t \mod n}$
7 $R_i[j] \leftarrow \sum_{t=0}^{T} r_i^t[j]$
8 send $R_i[j]$ to client
9 **for** $t \leftarrow T$ **to** 1 **do**
10 receive $r_{i-t \mod n}^t[j]$ from server $p_{i-t \mod n}$
11 $S_i[j] \leftarrow \begin{cases} \sum_{t=0}^{T} r_{i-t \mod n}^t[j] & \text{if} D_i[j] = 0 \\ \text{new random number} & \text{else} \end{cases}$
12 send $S_i[j]$ to client
13 **for** client **do**
14 $Y[j] \leftarrow \begin{cases} 1 & \text{if} \sum_{i=1}^{n} (R_i[j] - S_i[j]) = 0 \\ 0 & \text{else} \end{cases}$

3.2 Client/Vertical-Servers Set Intersection Protocol

We provide a special variant of a private set intersection as a major building block of both our model training and classification protocol. Assume that each vertical server p_i contains a binary vector $D_i \in \{0,1\}^s$, where $D_i[k] = 1$ if p_i supports the element k, and $D_i[k] = 0$, otherwise. The output is a binary vector $Y \in \{0,1\}^s$ with

$$Y[j] = \prod_{i=1}^{n} D_i[j]. \tag{1}$$

No vertical server p_i is allowed to reveal any value of D_j, $j \neq i$ or $f(c)$. Let T be a collusion threshold parameter. If less than T servers collude with each other, they can only exchange their inputs, but cannot induce input values of any other server or $f(c)$. A client cl receives the output, but should not learn anything else. Algorithm 1 solves this problem and calculates each $Y[j] \in Y$ independently from the others via a zero-sharing method. In zero-sharing, the servers distribute random numbers that sum up to zero. Then, every p_i sends its shares to cl if $D_i[j] = 1$.

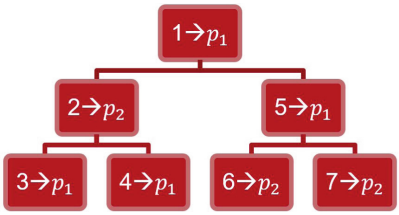

Fig. 1. Example of a random tree skeleton.

Table 1. Example of privately labeled tree.

Vertical server	Node	Attribute
p_1	1	Outlook
p_1	3, 4, 5	Humidity
p_2	2, 6, 7	Overcast

Algorithm 1 explains the steps of checking whether an element d is supported by all vertical servers in detail. All arithmetics are modulo integer operations in a sufficiently large field with the bit length b and all random numbers are uniformly distributed within this field. First, every p_i generates $T+1$ random numbers, $\{r_i^0, \ldots, r_i^T\}$, where $T \in [1, n-1]$ (lines 3–5), sends their sum to cl (lines 7–8) and scatters $\{r_i^1, \ldots, r_i^T\}$ to T other servers (line 6 and 10). Then, every p_i sends the sum of $r_i^0[j]$ and all received values to cl, if it holds d; otherwise, it sends a random number (lines 11–12). The client cannot distinguish between this random number or the sum, which is composed of other random numbers. The client adds up all values it received in the first round and subtracts all the values that were received in the second round (line 14). If the result is zero, d is not held by all servers with certainty, but in case the results do not sum up to zero, all the servers hold d with a probability of $1 - 1/2^b$. The probability of a false positive is therefore negligible. For our classification purposes, the size of an intersection should be always one (see Sects. 3.3 and 3.4). Hence, the occurrence of a false positive can be detected easily and the procedure restarts for the candidate elements with new random seeds.

3.3 Building a Privacy-Preserved Random Forest

Algorithm 2 presents the training steps of a private random forest. The model is distributed over all vertical servers and a commodity server. Each one receives the same tree skeletons similar to the model already proposed by other authors [23, 26] (lines 1–3). It maps an identifier and a party p_i to each branch in a random assignment process. Every server maps selected attributes to each node randomly (lines 4–6). Figure 1 illustrates a sample tree skeleton and Table 1 indicates its privately labeled tree.

As the tree is built randomly, no data records have been used yet. In the following steps, only the commodity server cs learns the class value distributions

Algorithm 2. Client/Vertical-Servers Random Forest Training

input : An attribute vector $X_i[j]$ for each training instance $j \in \{1, ..., m\}$ of each vertical server $\mathsf{p}_i, i \in \{1, ..., n\}$,
a collusion threshold parameter T,
a commodity server cs,
a server p_c, that holds the class values $C[j] \in \mathrm{C}$,
the number of trees in a forest o

output: A mapping $M : \mathbb{N} \mapsto \mathrm{C}$, that maps the index of each leaf node to the most associated class value

```
1  for one arbitrary server do
2      for k ← 1 to o do // for each tree of random forest do
3          tree^k ← new RandomTreeSkeleton()
4  for k ← 1 to o do // for each tree of random forest do
5      foreach vertical server p_i do
6          tree_i^k ← p_i.labelPrivately(tree^k)
7  for j ← 1 to m do // for each record do
8      for k ← 1 to o do // for each tree of random forest do
9          foreach vertical server p_i do
10             D_i ← p_i.getAllCandidateLeafs(X_i[j], tree_i^k)
11         leafID ← CVSSI({D_i}, T)
12         cs.store(leafID, p_c.getClassValue(C[j]))
```

of the leaves. It receives an assignment of a leaf node to each instance for each decision tree of the random forest. As long as *cs* does not collude with any vertical server, it cannot associate any attribute or class label with the identity of any instance. Algorithm 1 generates the leaf node that is provided to *cs* and the commodity server is treated as a client in this context. First, a preprocessing step is executed and each p_i assigns $D_i[l]$ with a one if the instance j reaches the leaf l (based on the attributes in $X_i[j]$), or a zero if not (line 10). Let $\{D_i\}$ denote the collection of private vectors D_i of all p_i. We get the output of Algorithm 1 for the record j, which is the leaf ID, $leafID$ (line 11). Then the commodity server updates the class distribution statistics of $leafID$ with $f(j)$ which is received from the class server p_c (line 12).

3.4 Client/Vertical-Servers Random Forest Classification

To classify a test instance c for a client whose attributes are stored at the vertical servers, we apply Algorithm 3. For that, all vertical servers p_i need to initialize a vector D_i with one at $D_i[l]$ if c reaches the leaf l or with zero, otherwise (line 3). Client c conforms to cl in Algorithm 1, so it receives the $leafID = c.leafIDs_{tree}$ corresponding to its attributes for each decision tree (line 4). Subsequently, it sends a request with all the leaf IDs to the commodity server to receive the most likely class label.

Algorithm 3. Client/Vertical-Servers Random Forest Classification

input : an attribute vector $X_i[c]$ of each vertical server $\mathsf{p}_i, i \in \{1, ..., n\}$,
a collusion threshold parameter T,
a commodity server cs with a mapping $M : \mathbb{N} \mapsto$ C, that maps the index of each leaf node to the most associated class value,
a client c,
an ensemble of $o \times n$ decision trees $\{\mathsf{tree}_i^k\}$

output: Classification of c

1 **for** $k \leftarrow 1$ **to** o **do** `// for each tree of random forest do`
2 **foreach** vertical server p_i **do**
3 $D_i \leftarrow \mathsf{p}_i$.`getAllCandidateLeafs`$(X_i[c], \mathsf{tree}_i^k)$
4 c.leafIDs $_{\mathsf{tree}} \leftarrow$ `CVSSI`$(\{D_i\}, T)$
5 c.classValue $\leftarrow$ cs.`classify`(c.leafIDs)

Note About the Client-Commodity Server Communication: The communication between c and cs is straightforward. Note that cs can read the client's request in clear text, but the client can communicate with the commodity server anonymously, so that cs cannot link the request with any other sensitive data or the identity of c. This communication can be done via a string-select oblivious transfer protocol, so that the commodity server does not learn the input of the client ($leafIDs$) and the output of the protocol. Kolesnikov *et al.* [17] provide an efficient 1-of-n oblivious transfer protocol, which can be applied here. It requires roughly four times the costs of a 1-out-of-2 oblivious transfer in an amortized setting and, therefore, is highly scalable. Moreover, c should get a shared one-time-password by one or more parties to prevent it from sending multiple malicious requests to cs, and not to be able to deduce sensitive information about the model and the underlying data. If these passwords are generated by the vertical servers, cs cannot associate them with individual clients even if the communication is not oblivious.

3.5 Security Analysis

In this section, we analyze the robustness of Algorithms 1–3 to information leakage. In Algorithm 2 (line 11) and 3 (line 4), the interactions among servers are limited to the interactions in Algorithm 1, hence, the security aspects of Algorithm 1 are directly transferable to them. Assuming that cs does not collude with any other server, the leaf IDs, class labels and input of vertical servers are secure against semi-honest and malicious attacks. The security level of the communication between the client and the commodity server (line 5 of Algorithm 3) is adaptable as discussed in Sect. 3.4. Here, we will discuss different security aspects of Algorithm 1:

– **Disclosure of the output:** The goal of Algorithm 1 is that the vertical servers input their private sets and the commodity server receives the inter-

section as the output. Assume that one or more parties try to reveal information about the output. The only messages they receive are random based zero-shares of other servers, which are independent from both their own input and any input of other parties. Consequently, even malicious parties have no opportunity to disclose anything about the output.

– **Association of input and output:** In case of a collusion between the commodity server and a vertical server, the collaborators can associate the identities of all the training records to their corresponding sensitive class values and leaf nodes, and therefore, similarities between the records as well. That is the reason for the requirement of having a trusted commodity server, which does not collude with any vertical server. Despite this restriction, using a commodity server improves the runtime effectively, and – according to [10] – finding such a *cs* is feasible in practice. It makes no difference if the *cs* is semi-honest (also known as honest-but-curious) or malicious, because it acts only as a receiver in the training mode and receives only unconditional messages that it cannot manipulate by own messages in the classification mode.
– **Disclosure of the input of other parties:** In the multi-party setting, there is a general risk of collusion between the data holding parties to combine their input data maliciously in order to violate an individual's privacy. However, this risk exists independent of the data mining protocols, hence it cannot be prevented in their design. As a protocol dependent aspect, we consider a case where b colluding vertical servers try to reveal the input data of one or more other servers. Looking into the Algorithm 1 indicates that there is no difference between semi-honest and malicious behaviour again. In the first part of algorithm (lines 3–8), every vertical server distributes numbers independent of each other. In the second part (lines 9–12), $S_i[j]$ comprises either $r_i^0[j]$ or another random number but no direct input data besides the numbers of other parties. This procedure happens independent of the messages of other parties, and consequently the public input of any party ($S_i[j]$) does not reveal any information if the numbers sent by other parties are generated randomly or with a malicious intention.

One adversary might try to find out whether an element d is supported by all parties or a particular vertical server p_x. The question if all parties hold d is defined by $\sum_{i=0}^{n}(R_i - S_i)$. Since each $S_i[j]$ is directly sent to the commodity server, only the commodity server (or all p_i together) is able to learn it. In order to find out whether a particular vertical server p_x holds d, adversaries have to know if:

$$S_x = \sum_{t=0}^{T} r_{x-t}^t \mod n \Leftrightarrow S_x = \sum_{t=1}^{T} r_{x-t}^t \mod n + r_x^0. \tag{2}$$

The only exception is if all p_i support d, because in that case, it is trivial that a particular vertical server does it too. S_x is only known by p_x and the commodity server. Given a random S_x, it cannot be calculated from other values. Arranging the vertical servers in a cycle in clockwise direction, $\sum_{t=1}^{T} r_{x-t}^t \mod n$ can only be calculated by the T servers on the right side of

p_x. r_x^0 is only known by p_x or can be calculated from $R_x - \sum_{t=1}^{T} r_x^t$, where R_x is only known by p_x and commodity server and $\sum_{t=1}^{T} r_i^t$ can only be calculated by the T servers on the left side of p_i. In conclusion, at least $\min(n-1, 2T)$ colluding vertical servers and the commodity server are necessary to find out whether a particular party p_x supports an element d.

4 Experimental Results and Complexity

4.1 Experiment Settings and Datasets

We implemented and tested the main random forest framework in Java, with four versions of the private set intersection:

1. The CVSSI protocol as designed in Algorithm 1.
2. Du02 version where we used a modified version of the scalar product protocol by Du and Zhan [10], such that the commodity server receives the output. This version has the constraint of a commodity server like our CVSSI protocol and is very fast, but can only be used for two-party problems.
3. A simple asymmetric public-key encryption scheme (Paillier encryption) that Vaidya and Clifton first used in the distributed decision tree context [8,26], because it fulfills the requirement of additive homomorphy. One party encrypts the identifier of d if it supports the element d else a zero with the public key. Then, each vertical server multiplies the encrypted intermediate result of its predecessor with a one if it supports d and with a zero if it does not. At the end, the results for each d are summed up, so that the total result is the encryption of d, because in our case only one d is supported by all parties together. Only the commodity server has the private key and can decrypt the result of the last vertical server and gets either a one or a zero. For simplicity, we do not use the state-of-the-art Paillier encryption, but give an idea of homomorphic encryption techniques.
4. A procedure with public splits like by Suthampan and Maneewongvatana [23] instead of a private set intersection method as a baseline, which is very straightforward, but reveals information we want to protect.

All experiments were executed on a single device with a dual core intel i7-5500U cpu and a 8 GB RAM. For the current results, we did not use a framework to simulate bandwidth and latency of a network of different devices. We tested the scalability on different real-world datasets of the UCI Machine Learning Repository with different parameters: number of vertical servers n, the collusion threshold parameter T, and the number of leaf nodes. The number of leaf nodes is β^δ, if the tree depth δ and the number of splits in a branch β are fixed in our experiments.

Table 2. Communication costs

Intersection protocol	2 vertical server	>2 vertical server
Du02	$4\,\mathrm{s} * b$	–
CVSSI	$6\,\mathrm{s} * b$	$s * n * (T + 2) * b$
Paillier encryption	$2\,\mathrm{s} * B$	$s * n * B$
Non private	–	–

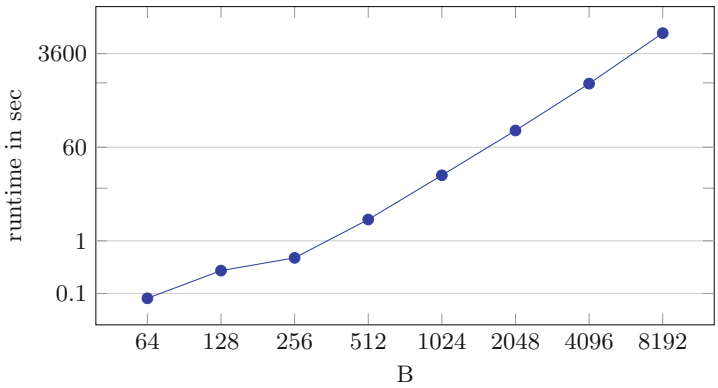

Fig. 2. Runtime of Paillier set intersection with three parties and vector length 1000.

4.2 Complexity Comparison

The computational complexity of Algorithm 1 is $O(s \times n \times T)$, with the vector length s, the number of vertical servers n and $T < n$. The algorithm Du02 requires $O(s)$ elementary operations, which is of the same order in a two-party setting. The Paillier encryption version consumes $(s+1)n$ bit multiplications for the encryption and summation and one bit exponentiation for the decryption. The fourth, not private version does not use a set intersection protocol. Instead, one responsible party broadcasts the supported records at each branch and leaf node. Hence, there are no computation and communication costs for a set intersection computation. Table 2 shows the communication costs depending on the bit length b of a data type and B as the bit length of a public key.

Apart from some initialization costs, Algorithm 2 calls $m \times o$ times a private set intersection protocol with the input size β^δ. Before the set intersection can be executed, every vertical server has to do the preprocessing step of filling the input vectors with a complexity of $O(\beta^\delta)$. This equates to a total computation and communication complexity of $O(m \times n \times o \times T \times \beta^\delta)$ in connection with Algorithm 1 (CVSSI). The total complexity of the non-private version is $O(m \times n \times o \times \beta^\delta)$. This is because all supported records are broadcast to every vertical server at each tree node once. In the deeper nodes, the number of supported instances is much smaller than m.

4.3 Performance Analysis

Figure 2 visualizes the exponential dependency of the runtime of homomorphic encryption schemes in relation to key bit length B. The German federal office for information security recommends a key length of 2,000 to 3,000 bits [7], which leads to a runtime of a few minutes for a single small vector with 1,000 elements in [26] and our experiments. This is rather infeasible for the whole tree building and classification procedure. Our CVSSI algorithm requires 1 ms for this task. The recently published private set intersection protocol by Kolesnikov *et al.* [17] runs also in less than a second in their environment and might be an alternative building block. However, one has to consider that the authors in [17] used a much more elaborate framework to simulate communication costs than we did.

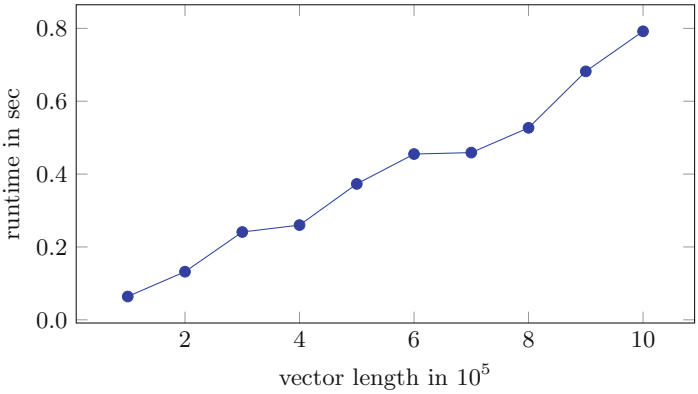

Fig. 3. Runtime of CVSSI (number of vertical servers: 4, T = 3, 100 repetitions).

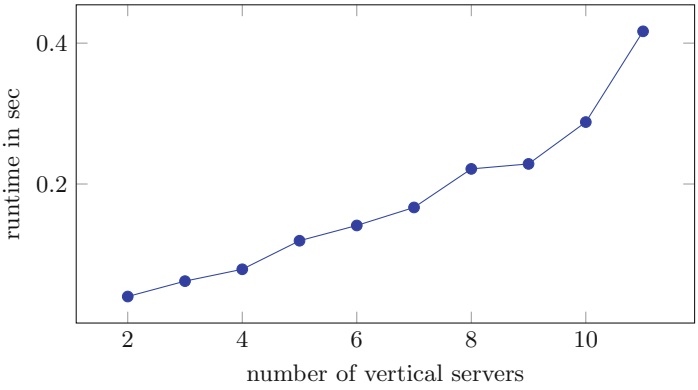

Fig. 4. Runtime of CVSSI (vector length: 10^6, $T = 3$).

Figures 3 and 4 confirm the linear scalability of our CVSSI algorithm in terms of n and o. We obtain similar results for T. They also show that the

protocol is feasible for larger vector sizes and a higher number of involved servers. Table 3 contrasts the runtimes of the four versions on the small car dataset (1728 instances) with a small number of trees (5) in a two vertical server setting. 90% of the instances are used for training and 10% for testing. As expected, the Paillier encryption version requires several minutes, although we set B to the unacceptably low value of 64. Our approach, the CVSSI protocol, takes less than half a second per tree. The Du02 version performs up to six times faster than our approach, which might be due to more effective vector operations in our implementations. The variant without private set intersection suggests further potential for improvement, but suffers from the inference problem. Table 4 shows the runtime of CVSSIP on real-world data sets with five parties and 20 randomly generated trees in seconds. This suggests that the approach is feasible in practice.

Table 3. Runtime on the car dataset, with two parties, 4 splits per branch, tree depth 5, in seconds and 10 random trees.

Trees	Paillier enc.	CVSSIP	Du02	Without PSI
1	115.519	0.464	0.080	0.003
2	238.991	0.922	0.169	0.006
4	461.899	1.796	0.321	0.008
6	693.079	2.701	0.466	0.011
8	919.099	3.591	0.639	0.014
10	1,165.374	4.631	0.833	0.015

Table 4. Runtime of CVSSIP on UCI ML data sets with five parties and 20 random trees.

Dataset	n	β^δ	runtime (s)
Cars	1,728	4^6	155.7
Contraceptive	1,473	2^9	16.8
Hepatitis (no missing values)	80	2^{17}	11,507.4
Nursery	12,960	5^7	22,783.6
Phishing websites	11,055	2^{13}	16,231.7
Thoraric surgery	470	2^{13}	173.1

5 Conclusion

We presented a new architecture which is a hybrid approach of private evaluation and classification on vertically partitioned data. This setting might become more

interesting in the future with the increasing use of private data and collaborations of companies, governments and different organizations. We provided a closed, lightweight and feasible solution with adaptable security levels. Additionally, it is highly parallelizable. The main drawback is the assumption of a central non-colluding commodity server. Making use of the results of Kolesnikov *et al.* on 1-out-of-n oblivious transfer and private set intersection [16,17], it may be possible to overcome this dependency in the future.

References

1. Aggarwal, C.C., Yu, P.S.: A general survey of privacy-preserving data mining models and algorithms. In: Aggarwal, C.C., Yu, P.S. (eds.) Privacy-Preserving Data Mining, pp. 11–52. Springer, Boston (2008). https://doi.org/10.1007/978-0-387-70992-5_2
2. Blockeel, H., Raedt, L.D.: Top-down induction of first-order logical decision trees. Artif. Intell. **101**(1), 285–297 (1998)
3. Bojarski, M., Choromanska, A., Choromanski, K., Lecun, Y.: Differentially- and non-differentially-private random decision trees, October 2014
4. Bost, R., Popa, R.A., Tu, S., Goldwasser, S.: Machine learning classification over encrypted data. Cryptology ePrint Archive, Report 2014/331 (2014)
5. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
6. Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS), pp. 498–507 (2007)
7. Bundesamt für Sicherheit in der Informationstechnik: Kryptographische verfahren: Empfehlungen und schluessellaengen, May 2018. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile&v=8
8. Damgård, I., Jurik, M., Nielsen, J.: A generalization of paillier's public-key system with applications to electronic voting. Int. J. Inf. Secur. **9**, 371–385 (2003)
9. Drummond, C., Holte, R.C.: Exploiting the cost (in)sensitivity of decision tree splitting criteria. In: International Conference on Machine Learning (ICML), pp. 239–246 (2000)
10. Du, W., Zhan, Z.: Building decision tree classifier on private data. In: Proceedings of the IEEE International Conference on Privacy, Security and Data Mining (CRPIT)-Volume 14, pp. 1–8 (2002)
11. Emekci, F., Sahin, O., Agrawal, D., Abbadi, A.E.: Privacy preserving decision tree learning over multiple parties. Data Knowl. Eng. **63**(2), 348–361 (2007)
12. Friedman, A., Schuster, A.: Data mining with differential privacy. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 493–502 (2010)
13. Wu, D.J., Feng, T., Naehrig, M., Lauter, K.: Privately evaluating decision trees and random forests. Proc. Priv. Enhancing Technol. **2016**, 335–355 (2016)
14. Jagannathan, G., Pillaipakkamnatt, K., Wright, R.N.: A practical differentially private random decision tree classifier. In: IEEE International Conference on Data Mining Workshops, pp. 114–121, December 2009
15. Kaghazgaran, P., Takabi, H.: Differentially private decision tree learning from distributed data, May 2015

16. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: Proceedings of the ACM SIG SAC Conference on Computer and Communications Security (CCS), pp. 818–829 (2016)
17. Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M., Trieu, N.: Practical multi-party private set intersection from symmetric-key techniques. In: Proceedings of the ACM SIG SAC Conference on Computer and Communications Security (CCS), pp. 1257–1272 (2017)
18. Lindell, P.: Privacy preserving data mining. J. Cryptol. **15**(3), 177–206 (2002)
19. Liu, X., Li, Q., Li, T., Chen, D.: Differentially private classification with decision tree ensemble. Appl. Soft Comput. **62**, 807–816 (2018)
20. Ma, Q., Deng, P.: Secure multi-party protocols for privacy preserving data mining. In: Li, Y., Huynh, D.T., Das, S.K., Du, D.-Z. (eds.) WASA 2008. LNCS, vol. 5258, pp. 526–537. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88582-5_49
21. Quinlan, J.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
22. Sravya, C.L., Lakshmi, G.R.: Privacy-preserving data mining with random decision tree framework. IOSR J. Comput. Eng. **19**, 43–49 (2017)
23. Suthampan, E., Maneewongvatana, S.: Privacy preserving decision tree in multi party environment. In: Lee, G.G., Yamada, A., Meng, H., Myaeng, S.H. (eds.) AIRS 2005. LNCS, vol. 3689, pp. 727–732. Springer, Heidelberg (2005). https://doi.org/10.1007/11562382_75
24. Tai, R.K.H., Ma, J.P.K., Zhao, Y., Chow, S.S.M.: Privacy-preserving decision trees evaluation via linear functions. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) ESORICS 2017. LNCS, vol. 10493, pp. 494–512. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66399-9_27
25. Teng, Z., Du, W.: A hybrid multi-group privacy-preserving approach for building decision trees. In: Zhou, Z.-H., Li, H., Yang, Q. (eds.) PAKDD 2007. LNCS (LNAI), vol. 4426, pp. 296–307. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71701-0_30
26. Vaidya, J., Clifton, C.: Privacy-preserving decision trees over vertically partitioned data. In: Jajodia, S., Wijesekera, D. (eds.) DBSec 2005. LNCS, vol. 3654, pp. 139–152. Springer, Heidelberg (2005). https://doi.org/10.1007/11535706_11
27. Zhan, J.Z., Chang, L.W., Matwin, S.: Privacy-preserving multi-party decision tree induction. In: Farkas, C., Samarati, P. (eds.) DBSec 2004. IIFIP, vol. 144, pp. 341–355. Springer, Boston, MA (2004). https://doi.org/10.1007/1-4020-8128-6_23