Master Thesis

Privacy Preserving Client/Vertical-Servers
Random Forest-based Classification

Derian Huibert Boer

1. Reviewer  Prof. Dr. Stefan Kramer
Institute of Computer Science
Johannes Gutenberg University Mainz

2. Reviewer  Prof. Dr. Ernst Althaus
Institute of Computer Science
Johannes Gutenberg University Mainz

Supervisors  Zahra Ahmadi and Prof. Dr. Stefan Kramer

August 9th, 2018
Master Thesis
Title: Privacy Preserving Client/Vertical-Servers Random Forest-based Classification
Submission date: August 9th, 2018
Reviewers: Prof. Dr. Stefan Kramer and Prof. Dr. Ernst Althaus
Supervisors: Zahra Ahmadi and Prof. Dr. Stefan Kramer

This thesis is submitted in partial fulfillment of the requirements for the degree
Master of Science (M.Sc.) in Business Informatics to:

Johannes Gutenberg University Mainz
Institute of Computer Science
Faculty 08: Physics, Mathematics and Computer Science
Staudingerweg 7
55128 Mainz

Derian Huibert Boer
Abstract

This thesis presents a novel client/vertical-servers architecture for a hybrid multi-party classification problem. The model consists of clients whose features are distributed on multiple servers and remain secret during learning and classification. As a use-case example, 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. Other applications can be found in spam filtering, crime reduction, credit assessment or clinical diagnosis.

The solution of this thesis builds a distributed random forest and labels its leaves via a special private set intersection protocol. This protocol provides a central commodity server with anonymous conditional statistics of the leaves, but does not disclose any other information to any participant. Subsequently, the private set intersection protocol can be used to classify new clients in combination with the commodity server’s statistics in a privacy-preserving way. 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.

In summary, this thesis presents a client/vertical-servers set intersection protocol, a training and a classification algorithm and provides run-time results on some real-world data sets that demonstrate its beneficial linear scaling properties and feasibility. The thesis covers different security aspects, offers some notes about how the core architecture can be extended to fulfill additional requirements, and gives an outlook on further advancement opportunities.

Keywords: vertically partitioned data, private evaluation, secure multi-party computation, privacy preserving data mining, random forest
# Contents

1 Introduction 1  
   1.1 Background and Contribution 1  
   1.2 Thesis Structure 2  

2 Privacy Preserving Decision Tree Classification 5  
   2.1 Commonly used Decision Tree Algorithms 5  
   2.2 Privacy Definitions 8  
   2.3 Types of Privacy Preserving Data Mining Methods 10  
   2.4 Private Set Intersection 11  
   2.5 Decision Tree Algorithms for Distributed Data 14  

3 Client/Vertical-Server Set Intersection 21  
   3.1 Problem Statement 21  
   3.2 Algorithm 22  
   3.3 Security Analysis 23  
   3.4 Complexity Analysis 25  

4 Random Forest Computation in a Client/Vertical-Servers Setting 27  
   4.1 Problem Statement 27  
   4.2 Random Forest Training Protocol 28  
   4.3 Random Forest Classification Protocol 30  
   4.4 Security Analysis 31  
   4.5 Complexity Analysis 32  
   4.6 Extension Notes 33  

5 Experimental Evaluation 37  
   5.1 Experiment Settings and Datasets 37  
   5.2 Experimental Results 38  

6 Conclusion and Future Work 45  

Bibliography and Websites 47
Introduction

In the past, many data mining and machine learning methods for data processing in both research and business applications have been established. Well-known examples are text and speech recognition, gene expression analysis, financial prediction and autonomous driving. 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 increasing. In this context, the goal is to make use of the potential knowledge that additionally measured or mined data can provide. At the same time, the challenges of preserving privacy of personal and other sensitive data grows. 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.

1.1 Background and Contribution

Aspects of privacy-preserving data mining 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 affect the effectiveness. In SMC, the data sets remain completely private and hence, do not need to 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 its 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.

This thesis considers a special case of the aforementioned private evaluation scenario...
for decision tree-based classification and set intersection in the SMC scenario. In this case, the data of some clients is vertically partitioned and distributed across multiple servers. It is sufficient if at least one server knows the class labels of the training instances. Each party sees only its own 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. No other party is able to learn any leaf node statistics. 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 such that the client gets a shared sum of its leaf node identifier from each party. No server learns other features or information about the output at this step.
- Then, the client can anonymously ask the commodity server about the class value statistics of its leaf node.

There are several applications for this framework in spam filtering, crime reduction or credit assessment. For example, 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 wants to disclose its information to the others. This thesis elaborates this architecture and algorithms in details and provides a closed, lightweight and feasible solution with adaptable security levels. Moreover, it gives an overview of the background and evaluates experimental results to assess its feasibility.

1.2 Thesis Structure

This thesis is structured in the following way:

Chapter 2
In Chapter 2, important basics and notions are defined. It starts with an overview of the C4.5 and random forest algorithms as two well known and commonly used representations of decision tree building methods. Then, the chapter goes over in privacy preserving data mining (PPDM), beginning with the definition of two basic privacy levels. This is followed by an overview of the typical PPDM models like randomization and SMC. The last subsection provides a description of horizontally and vertically partitioned datasets, the client/server model in matters of PPDM and several representative suitable classification approaches.
Chapter 3
Chapter 3 introduces the client/vertical-servers architecture, which is a hybrid of two typical data partitioning models in PPDM. Furthermore, it provides a simple private set intersection algorithm, the client/vertical-servers set intersection protocol (CVSSIP), as an important tool for the following classification architecture, and discusses its robustness against different types of attacks and compares its theoretical complexity to some other private set intersection protocols.

Chapter 4
In Chapter 4, the client/vertical-servers architecture is expanded to a classification problem setting, which permits a certain commodity server. It starts with a description of the considered scenario, then introduces a random forest training and an evaluation algorithm that handle this scenario, and provides a related security and complexity analysis. Finally, additional notes about possible extensions to the aforementioned algorithms are given.

Chapter 5
The concepts of the Chapters 3 and 4 are experimentally evaluated in the subsequent Chapter 5. Here, the experiment settings are explained and the used datasets are described at first. After that, the test results of both the CVSSIP and the client/vertical-servers random forest algorithms are visualized and interpreted.

Chapter 6
In Chapter 6 summarizes the most important aspects and test results of this thesis. Finally, I point out a few minor restrictions, which give motivation for further developments.
Privacy Preserving Decision Tree Classification

This chapter serves as basis for the new framework presented in the following chapters. It starts with two widely used non-private decision tree based classification algorithms and an introduction to definitions and methods of privacy preserving data mining in general. This is supplemented by an overview of some privacy preserving tree classification methods and a few private set intersection solutions as a possible building block of secure multi-party classification protocols.

2.1 Commonly used Decision Tree Algorithms

Decision trees are commonly utilized not only for solving classification and regression problems, but also for clustering with cluster descriptions [4].

A widely-used, intuitive decision tree algorithm is the ID3 algorithm [36] and its extension C4.5 [35]. 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 [13]. Decision trees have shown promising results on many problems. Nevertheless, their performance can be improved by the combination of the outcomes from many largely independent decision trees. Overfitting and variance can be reduced by bootstrap sampling and random feature selection in the tree building process, which leads to the random forest algorithm [8]. Random forest is a classification model that exists of more uncorrelated decision trees whose branches are grown by random attribute selections. This provides the advantage that trees can be constructed without any data queries in the first step. Although there are other classification methods like Bayesian networks, support vector machines or k-nearest-neighbors, this thesis focuses on a secure multi-party computation model for decision trees, as a successful and well-interpretable learning model. The rest of this section is about the C4.5 and random forest algorithms in general, whereas Section 2.5 deals with privacy preserving tree models on distributed data.
Algorithm 1: Recursive_ID3

C4.5

As with other decision tree classifiers (e.g. CART), the tree creation process of C4.5 consists of two phases. In the first phase (building phase), the tree grows from the root to the leaves by recursively splitting the training records until all or most of them that belong to one node are of the same class. The split is based on a locally optimal criterion on each node. The second phase extends the older ID3 algorithm by a pruning step, in which the leaves with a low support are cut off to improve generalization. Algorithm 1 shows the tree building phase of ID3 and C4.5. It is a recursive function that takes a set of records \( R \), a set of remaining attributes \( A \), whose cardinalities decrease at each iteration, and a certain attribute, the class attribute \( c \), as input. Initially, \( R \) is the set of all training records and \( A \) the set of all features. \( c \) is the attribute that shall be predicted by the algorithm’s output and remains fixed. At each iteration, the locally optimal split is performed with the attribute \( \alpha \) of \( A \) that has the highest information gain and therefore separates the remaining records in partitions \( P_i \) with the lowest impurities (line 6).

Let \( a \in A \) denote a concrete attribute, \( a_i \) be a discrete value of \( a \) and \( S(a_i) \) denote the set of all transactions whose value of \( a \) is equal to \( a_i \). Then, the information gain, considering the currently supported records \( R_i \), is calculated by the reduction of the impurity:

\[
\text{informationGain}(R, a, c) = \text{Impurity}(R, c) - \sum_{a_i \in a} \frac{|R \cap S(a_i)|}{|R|} \cdot \text{Impurity}(R \cap S(a_i), c).
\]

(2.1)

The impurity could be simply measured by the accuracy e.g., but because of its high sensitivity to costs, the accuracy might not be the appropriate measure to evaluate a classifier’s performance if the class frequencies are imbalanced, which is the usual
Splitting criteria \( f(p, n) \)  

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>( \min(p, n) )</td>
</tr>
<tr>
<td>Gini</td>
<td>( p^2 + n^2 )</td>
</tr>
<tr>
<td>Entropy</td>
<td>( p \log_2(p) + n \log_2(n) )</td>
</tr>
<tr>
<td>DKM</td>
<td>( \frac{\sqrt{pn}}{2m} )</td>
</tr>
</tbody>
</table>

Tab. 2.1: Impurity metrics for binary classes

case in real world applications. The most common (and original) splitting criteria is the (Shannon) entropy:

\[
\text{Impurity}(\hat{R}, c) = \sum_{c_i \in c} p(c_i) \log_2 p(c_i), p(c_i) = \frac{|\hat{R} \cap S(c_i)|}{|\hat{R}|}.
\]  

(2.2)

\( R \cap S(c_i) \) denotes the subset of \( R \) with all records that have the class value \( c_i \). Drummond and Holte [13] compare different impurity functions for binary classes. They are listed in Table 2.1 as functions \( f(p, n) \), where \( p \) is the fraction of \( R \) that has the positive class value and \( n = 1 - p \) is the counterpart. The cost sensitivity is not directly related to performance, hence they are all legitimate. Furthermore, they can be combined with cost-sensitive pruning methods. After an attribute \( \alpha \) has been chosen, \( R \) is split in a partition \( P_i = R \cap S(\alpha_i) \) for each value of \( \alpha \) (line 7). In the next step, a node \( \nu \) is labeled with \( \alpha \) and connected with the output of \( \text{Recursive}_{\text{ID3}}(P_i, A \setminus \alpha, c) \) via an edge, which is labeled with \( \alpha_i \). This is done for each \( \alpha_i \). Then, \( \nu \) is returned as the output (line 8). The function terminates if there are no more attributes left in \( A \) (line 1) or all transactions in \( R \) are of the same class (line 3) or \( |R| \) is below a user defined threshold (optional). If at least one of the conditions is fulfilled, the ID3 algorithm returns a leaf node, which is labeled with the class value that the majority of the records in \( R \) has.

Random Forest

A trained classifier can specialize strongly on the selection of the used training set and can include many rules that rely on only a few training instances and are not applicable in general. This is called overfitting and causes a high variance, an error from sensitivity to small fluctuations in the training set. In contrast to a single decision tree, a random forest can reduce the variance [8]. Such a forest is an ensemble of many uncorrelated decision trees that are grown with a certain kind of randomization. The testing is conducted via a majority voting of predictions of the separate trees in a classification case. In case of a regression, the output is the average value of the predictions. Thereby each tree’s vote has the same weight. Bosch et al. [6] additionally take the posterior probability of a class to reach a leaf into account. The first method of randomization in the building process is bootstrap aggregating, the equally-distributed random selection of records with replacement. The second method is the random selection of \( s \) features in each splitting point. In
other algorithms, finding the best split attribute requires a lot of data queries. The random selection of features has the positive side effect that the number of necessary queries is reduced enormously, because only $s$ of all possible attributes have to be compared. For classification with $|A|$ attributes, the default value for $s$ is $\lfloor \sqrt{|A|} \rfloor$ and for regression it is $\lfloor \sqrt{|A|}/3 \rfloor$ [21]. If $s$ is set to one, the splits can even be done without accessing any training records. In Section 4.2, this is exploited to avoid communication costs in the training phase of a multi-party case and in Section 5.2.2 it is evaluated how the parameter $s$ influences the classification performance.

Algorithm 2 shows an implementation of the random forest building procedure in combination with Algorithm 3. Besides $s$, a customizable parameter is the depth of each tree $\delta$. A random forest consists of $o$ unpruned trees that are usually flatter than classifiers with a single tree. Algorithm 3, the recursive function to grow a single tree, is called for $o$ times with an independent random selection with replacement of $n$ out of $n$ instances (lines 1-4 of Algorithm 2). The main differences to Algorithm 1 regarding the growing process of a tree are the termination condition of a fixed tree size (line 1) and the limitation to $s$ randomly chosen features that are considered in the evaluation process for selecting the locally optimal splitting attribute (lines 4-5). The independence of the trees in the training and testing phase make random forests highly parallelizable and speeds up the training process on an appropriate hardware.

### 2.2 Privacy Definitions

In this work, I distinguish between semi-honest and malicious parties in the context of privacy preserving communication and make use of a commodity server as an enabler of efficient algorithms.

**Semi-honest and malicious model**

By definition, semi-honest parties follow the protocol as designed, but they collect all information they receive during the communication and try to use it to reveal private
### Algorithm 3: Recursive_Random_Tree

```latex
\textbf{input}: Records } R, \text{ candidate attributes } A, \text{ a class attribute } c, \\
\text{ randomization parameter } s, \text{ depth } d \text{ of the current tree } \\
\textbf{output}: A decision tree \\
1 \textbf{if } d = 0 \textbf{ then} \\
2 \quad \text{return a leaf node with class value } c_i \\
3 \quad \text{arg max } |R \cap S(c_i)| \\
4 \textbf{else} \\
5 \quad \hat{A} \leftarrow \text{randomly select } s \text{ features out of } A \\
6 \quad \alpha \leftarrow \text{arg max}_{a \in \hat{A}} \text{ informationGain}(R,a,c) \\
7 \quad \text{Partition } R \text{ in } |\alpha| \text{ partitions } P_i = R \cap S(\alpha_i) \\
8 \quad \text{return a tree whose root is labeled with } \alpha \text{ and is connected to the} \\
9 \quad \text{subtree Recursive_Random_Tree}(P_i, A \setminus \alpha, c, s, d-1) \text{ for all } p_i \text{'s via} \\
10 \quad \text{an edge that is labeled with } |\alpha_i| \\
11 \textbf{end}
```

information of other parties. That’s why a semi-honest party is often also called an
honest-but-curious one or a passive one. If a party additionally deviates from the
protocol to manipulate its outcomes, it is called a malicious one. For example, it
could exchange a local input or abort the protocol prematurely. The malicious model
is stronger than the semi-honest one, because it additionally guarantees that there
is no information leakage even if a party behaves malicious. Goldreich provides an
extensive formal description of the semi-honest and malicious model and possibilities
how parties can deviate from a protocol [19].

### Commodity Server

One of the main reasons why multi-party protocols are often impractical in real
world applications is the ambition to achieve an ideal security and the resulting
strictness and bad efficiency. In practice, there might not be the need of perfect
security. According to Du and Zahn people will accept the risk of less secure but much
more efficient solutions [14]. For this reason Beaver introduced the commodity-
based cryptography. In his model, commodity servers "provide security resources to
clients but are not involved in the clients’ computations themselves" [3]. Commodity
servers have no knowledge about each other and assist the clients via single messages,
commodities, to establish secure computations on shared resources. Usually, the use
of commodity servers increases the performance of a multi-party computing instance
enormously. It is necessary that the commodity servers do not collude with any
party. Otherwise, the model simply becomes the conventional multi-party model. Du
and Zahn suggest that finding such a commodity server is very feasible in practice
[15].
2.3 Types of Privacy Preserving Data Mining Methods

There are different concepts to preserve privacy, which can be applied to different privacy goals and go along with different properties in terms of accuracy, run-time and level of security. In the following, some main groups of privacy preserving data mining models are explained:

- randomization,
- k-anonymity, l-diversity and t-closeness,
- downgrading application effectiveness and
- secure multi-party computation (SMC).

Randomization

The goal of randomization techniques is to mask the attribute values of individual records [2]. This can be necessary if the data owner intends to share parts of a database or a classifier with partners e.g. and the privacy of individuals about which they are collected might be harmed. Even despite anonymization steps, the identity of a record might be indirectly derivable from other unique properties if background information is available. In order to prevent risks like this, sufficiently large noise is added to the data. On the other side, aggregate distributions should remain stable to maintain a suitable training base and the training algorithms have to be able to handle this larger synthetic noise.

k-anonymity, l-diversity and t-closeness

An other way to face the possibility of implicit identification of records from public datasets is the k-anonymity model and its enhancements. Here, the granularity of a dataset is reduced until any given record corresponds to at least k other records in the data. This can be realized by generalization and suppression techniques for instance. Masking the identity of single records via k-anonymity can not completely protect their privacy because the assignment to a homogeneous group with k individuals can be sufficient to revealing sensitive attribute values that the group's records have in common [1]. The l-diversity model tackles this problem by requesting intra-group diversity of a least l well represented values for every sensitive attribute within each possible group [31]. Sometimes not all characteristics of an attribute are equally sensitive. If one considers a disease describing attribute e.g., a positive value might be much more confidential than a negative one. The t-closeness model is a further enhancement on the concept of l-diversity. t-closeness is given if the distance between a distribution of a sensitive attribute within each anonymized group and its global distribution does not deviate by more than a threshold t [27].
Downgrading application effectiveness

Even if a database remains private and direct accesses on it is prevented, the output of data mining applications or query processing may disclose sensitive information. As a countermeasure, either the data or the application’s output can be modified. Some methods to intended downgrading of the applications effectiveness are auditing and limiting the output of query answers, association rule hiding by distortion of outputs or blocking of certain queries, and downgrading the accuracy of classifiers [1].

Secure multi-party computing

The aforementioned methods to protect privacy cause some loss of effectiveness of the data mining tools that use the edited data. Secure multi-party computing (SMC) techniques do not suffer from the natural trade-off between information loss and privacy. In SMC, the data sets remain completely private and hence do not have to be disturbed. Instead, special cryptographic communication protocols allow two or more parties to obtain aggregate results from their combined data, but each of them does not learn any 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 subject area is also know as distributed privacy preservation, because the integrated data is partitioned on multiple parties, which protect their shares. The most important data partitioning models are presented in Section 2.5. A special subcategory of SMC is the client-server (or sender-receiver) environment, where the owner of a sensitive trained model delivers a classification in dependency of a client’s sensitive features as a service while neither the server nor the client is able to reconstruct the counterparty’s input. Additionally, the server does not learn the output of a computation. In Chapter 4, a novel hybrid of the client-server architecture and an other distributed data model is introduced and an appropriate decision-tree-based SMC framework is elaborated. The downside of SMC techniques in general is their high communication and computation costs, but they have been reduced continuously since their introduction.

2.4 Private Set Intersection

Some secure branching multi-party concepts for classification over distributed data use a secure multi-party dot product or a similar private set intersection protocol as an important subroutine to determine common conditional class distributions of potential or already grown tree branches. Like the first publications in the field of SMC classification [15, 45], the privacy concept of the classification algorithms in this thesis is based on it. All protocols explained in this section are SMC techniques and hence lead to precise results but suffer from different other drawbacks.
Two-party scalar product protocol
Du and Zahn [15] provide and use a "scalar product protocol" for two parties, Alice and Bob, that have one sensitive binary vector each. Let $D_A \in \{0, 1\}^s$ and $D_B \in \{0, 1\}^s$ denote this vectors and $D_A \times D_B$ denote their dot product

$$D_A \times D_B = \sum_{j=1}^{s} D_A[j] \ast D_B[j].$$

(2.3)

As the outcome of the procedure, Alice should privately receive a share $v_A$ and Bob should privately receive its complement $v_B$, such that $v_A + v_B = D_A \times D_B$. A commodity server, which is not able to learn anything about the input and output of Alice and Bob as long as it does not collude with one of the other parties, is used to initialize the protocol. The individual steps are:

1. The commodity server generates two randomly filled vectors $R_A$ and $R_B$.
2. The commodity server generates a random number $r_A$ and a number $r_B$, such that $r_A + r_B = R_A \times R_B$.
3. The commodity server sends $r_A$ and $R_A$ to Alice and $r_B$ and $R_B$ to Bob.
4. Alice sends $S_A = D_A + R_A$ to Bob and Bob sends $S_B = D_B + R_B$ to Alice.
5. Bob generates a random number $v_B$ and sends $t = S_A \times D_B + (r_A - v_B)$ to Alice.
6. Alice calculates $v_A = t - R_A \times S_B + r_A$.

Figure 2.1 presents in a simple example which variables are created in which step. This procedure works without a complex encryption technique but requires a reliable commodity server.

Asymmetric encrypted set intersection protocol
In their SMC decision tree algorithm [45], Vaidya and Clifton use a private cardinality of set intersection protocol [46] for the main part. But since the size of all combinations of intersection sets and the class counts for each tree node are disclosed by their standard solution, they append a multi-party dot product protocol, despite its low efficiency. It fulfills the same purposes as the above scalar product
protocol but supports $n \geq 2$ parties $p_i$ with a vector $D_i = \{0, 1\}^s$ each. The goals is to securely solve
\[
\sum_{j=1}^{s} \prod_{i=1}^{n} D_i[j].
\]
(2.4)

Their solution is based on the Paillier threshold encryption scheme [11] and goes without a commodity server. The Paillier encryption is an asymmetric additive and homomorphic cryptosystem. Thus in this cryptosystem, for any two numbers $a$ and $b$, a one-way public-key encryption function $\text{encr}_{pk}$ and a certain modulus multiplication operator $\odot$, the following applies
\[
\text{encr}_{pk}(a + b) = \text{encr}_{pk}(a) \odot \text{encr}_{pk}(b).
\]
(2.5)

Furthermore, the Paillier encryption has the property
\[
k \ast \text{encr}_{pk}(a) = \text{encr}_{pk}(k \ast a).
\]
(2.6)

In the threshold scheme, the private key has been divided into $k$ parts, such that no single party is able to decrypt intermediate results in deviation to the protocol on its own. In a nutshell, the algorithm consists of these steps:

1. The first party sets $E_1[j] = \text{encr}_{pk}(D_1[j])$.

2. The parties $p_i, i = [2, n]$, set $E_i[j] = \begin{cases} E_{i-1}[j] \ast 1 & \text{if } D_i[j] = 1 \\ \text{encr}_{pk_i}(0) & \text{else} \end{cases}$.

3. The parties jointly decrypt $\text{encr}_{pk}(\sum_{j=1}^{s} E_n[j])$.

In the second step, each $p_i$ multiplies the encrypted intermediate result of its predecessor with one if $D_i[j] = 1$ and else encrypts zero. Both operations rerandomize the intermediate result of the predecessor such that privacy of $p_i$ is preserved. The practical complexity of this procedure is evaluated in Chapter 5.2.2.

**Symmetric key encrypted set intersection protocol**

In 2017, Kolenikov et al. developed a private set intersection (PSI) protocol [26] that uses symmetric key primitives and performs significantly faster than those with public-key operations. It is secure in the semi-honest sense. Its main building block is the oblivious evaluation of a programmable pseudorandom function (OPPRF), which enables an efficient 1-out-of-n oblivious transfer protocol that requires only a constant time in amortized setting [25]. In a 1-out-of-n oblivious transfer, a sender transfers one of out $n$ data base elements of information to a receiver, but remains oblivious which piece has been transferred. Section 4.6.2 includes more details about this oblivious transfer method. The PSI protocol consists of two major phases:
the conditional zero-sharing phase and the conditional reconstruction phase. In the first one, the parties jointly generate additive shares of zero, which means that every party obtains one private pseudo random number and all numbers together sum up to zero. In the second phase, each party \( p_i \) programs an instance of an OPPRF to output its share if \( D_i[j] \) is one else anything, for each element \( j \) that might be part of the performed intersection. If \( j \) is part of the intersection, the outputs of all OPPRF have to sum up to zero. In Chapter 3, a similar procedure is used for a novel set intersection protocol, which does not need an OPPRF or an other oblivious transfer method. Instead only a central commodity server receives the joint output (either the zero-shares or random outputs).

2.5 Decision Tree Algorithms for Distributed Data

In the multi-party computing scenario, data can be partitioned among parties vertically, where the parties have different attributes of equal data objects, or horizontally, where the parties have different data objects of a compatible structure. For simplicity, it can be assumed that the instances are stored in the same order along all parties in the horizontal scenario. Otherwise, they can be rearranged in a preprocessing step. In practice, these partitioning variants might overlap.

The client-server setting, which is better known as privacy-preserving evaluation, was developed later in this context. Although the starting situation is different, it is somewhat related to the vertical and horizontal partitioning models and is handled with similar solution approaches. In the privacy-preserving evaluation, a server has a sensitive model and a client has a sensitive feature vector as input. The aim is to classify the client’s data while the sensitive inputs (model and query) remain hidden from the counterparty. Figure 2.2 provides an example for each of the three mentioned partitioning models.

![Data partitioning models](image)

Fig. 2.2: Data partitioning models: (a) vertically partitioned data, (b) horizontally partitioned data; (c) privacy preserving evaluation model

The following subsections give a brief overview of the secure multi-party literature in the context of decision tree learning and evaluating.
### 2.5.1 SMC Protocols on Vertically Partitioned Data

The first decision tree algorithm on vertically partitioned data for two parties is a ID3-modification proposed by Du and Zahn [15]. In order to count the records that support a particular attribute or class value, which is the critical step of an entropy calculation, 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. The secure shared two-party scalar product protocol of Section 2.4 is applied to this binary vectors to get the conditional counts and therefrom obtain the conditional distributions after potential splits. Let \( p_A^c \) and \( p_B^c \) denote the shares of a conditional proportion of the class label \( c \in C \). The authors additionally present a logarithm protocol that outputs a shared sum \( v_A^c \) to Alice and \( v_B^c \) to Bob, such that
\[
\log(p_A^c + p_B^c) = v_A^c + v_B^c.
\]
By the the use of both protocols, the conditional entropy can be transformed into four dot products again:
\[
\sum_{c \in C} (p_A^c + p_B^c) \cdot \log(p_A^c + p_B^c) = \sum_{c \in C} (p_A^c + p_B^c) \cdot (v_A^c + v_B^c)
= \sum_{c \in C} (p_A^c) \cdot (v_A^c) + \sum_{c \in C} (p_B^c) \cdot (v_B^c) + \sum_{c \in C} (p_A^c) \cdot (v_B^c) + \sum_{c \in C} (p_B^c) \cdot (v_A^c)
= p_A \times v_A + p_B \times v_B + p_A \times v_B + p_B \times v_A.
\]
These protocols are lightweight solutions, but require a commodity server that must not collude with any of the parties. This approach is hardly extensible to a \( n \)-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 [45, 42] use trees whose nodes are mapped to the attributes that are only visible for the corresponding party. Vaidya and Clifton propose a multi-party dot product protocol and the less leaking secure private set intersection protocol [45] that is explained in Section 2.4 and can be applied in a similar way as the scalar product protocol of Du and Zahn [15]. In an approach of Suthampan and Maneewongvatana [42], each party calculates 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. By the use of this method, the communication costs are comparable with those of a non-private procedure on vertically distributed data. This approach is only feasible if the class values are available for all parties. The other downside of this approach is that the similarities among different records can still be leaked. Both the protocol of Vaidya and Clifton and the one of the latter approach support two or more parties.
The challenge of horizontally partitioned data, where parties have coincident attributes, is to sum up the conditional record counts of all parties without disclosing the counts of an individual party. This is done in order to get more accurate splits and leaf statistics by the combination of more training records.

As an initial work, Lindell and Pinkas [28] propose to employ Yao’s protocol for two parties [49] to find the attributes with the highest local information gain without sharing the conditional record counts explicitly and without using a trusted third party. Yao’s protocol solves his millionaire’s problem where two persons want to find out who is the richer one without disclosing any additional information. In Yao’s solution, the persons gain the result in a constant number of four rounds via the communication of garbled circuits. The problem is that the size of the circuits depends on the size of the input, which impairs the run-times of Lindell’s and Pinkas’ algorithm. Furthermore, Lindell and Pinkas provide a secure $x \ln(x)$-protocol to find the attribute with the lowest conditional entropy.

Emekci et al. [16] suggest an approach with lower communication and computation costs than the previous one [28]. It is predicated on a private summation protocol based on Shamir’s secret sharing [39]. The goal of Shamir’s secret sharing is to divide data $d$ (represented as an integer) into $n$ pieces in such a way that $d$ is easily reconstructable from any $k$ pieces, but $k-1$ pieces disclose no information about $d$. In the solution approach, a random polynomial $q(x)$ of the degree $k-1$ with the point $(0, d)$ is generated and $n$ data points $d_1, d_2, ..., d_n$ are distributed to the parties. Now, $d$ can be reconstructed with Lagrange polynomial interpolation on a two-dimensional plane. One computationally efficient approach to calculate $d$ is

$$d = \sum_{j=0}^{k-1} f(x_j) \prod_{m=0}^{k-1} \frac{x_m}{x_m - x_j}. \quad (2.8)$$

The protocol of Emekci et al. [16] consists of three phases: distribution phase, intermediate computation phase and final computation phase. In the first one, they treat the private counts as points of a polynomial. Each party distributes the bases of the corresponding polynomial function to the other parties. In the last phase, the functions are summed up and the total record sums can be derived from the resulting joint polynomial.

Later, Ma and Ping [30] combined the former idea [16] with the one by Suthampan and Maneewongvatana [42] for vertically distributed data and use the Gini index instead of the entropy as the impurity measure. Consequently, their ID3-framework can deal with both horizontally and vertically partitioned data.
2.5.3 Randomization Protocols

In some cases, a data owner wants to share data with certain partners or even in public and does not need to protect abstract views or trends of his data but wants to protect individual records. For example, cooperating companies might be willing to share their data but need to protect features of individual clients. In this case, applying strict SMC methods might be like taking a sledgehammer to crack a nut and using suitable randomization techniques could be the more appropriate solution.

One party functions as a data collector and the others send their randomized data for shared data mining purposes. Randomization approaches imply a trade-off between the individual’s privacy and the quality of the results but usually lead to faster results.

The first randomization based multi-party tree induction [50] used a multi-group randomized response (MRR) scheme for vertically partitioned data that works as follows: The attributes are partitioned into groups. In the first step, a user conducts a coin-flipping for each group and 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 costs of the tree building and the accuracy loss can be reduced by employing a hybrid of MRR and SMC [44]. The choice of an attribute for a local split is narrowed down to the \( \omega \) attributes that have the highest information gains on the combined randomized data. Afterwards, the final selection from the pre-selected attributes is done by an evaluation on the original data with a private dot product protocol.

Recently, many \( \epsilon \)-differential solutions for ID3 [18, 24] and tree ensembles [23, 5, 41, 29] were proposed. A solution is \( \epsilon \)-differential private if the outcome of a randomized calculation \( M \) is insensitive to any particular private value in the data set:

**Definition 1** A randomized computation \( M \) provides \( \epsilon \)-differential privacy, if for any datasets \( A \) and \( B \) with symmetric difference \( A \Delta B = 1 \), and any set of possible outcomes \( S \subseteq \text{Range}(M) \), \( \Pr[M(A) \in S] \leq \Pr[M(B) \in S] \times e^\epsilon \). [18]

The \( \epsilon \)-differential privacy model limits the amount of information that an adversary can gain about a particular record, even if he knows every other value in the database. [48] The randomization is obtained by the addition of noise, and the \( \epsilon \)-parameter can be seen as a "privacy budget". It fulfills the composability property, so the parameters of consecutive queries can be accumulated. In \( \epsilon \)-differential private ID3 algorithms, the stopping criteria should be adjusted to keep trees synthetically flat because the size of the subsets in deeper branches (leaves especially) of a tree are affected by the added noise at most. [18]

The major drawback, especially of the ID3 solutions, is the high variance in the
<table>
<thead>
<tr>
<th>Data set</th>
<th># records</th>
<th># queries</th>
<th>Accuracy orig. ID3</th>
<th>Acc. $\epsilon$-diff. private</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nursery</td>
<td>12960</td>
<td>176</td>
<td>98.19%</td>
<td>18.73%</td>
</tr>
<tr>
<td>Cong. Votes</td>
<td>435</td>
<td>97</td>
<td>94.48%</td>
<td>2.53%</td>
</tr>
<tr>
<td>Mushroom</td>
<td>8124</td>
<td>267</td>
<td>100%</td>
<td>1.48%</td>
</tr>
</tbody>
</table>

Table 2.2: Example of poor accuracy results of $\epsilon$-diff. private ID3 [23]

accuracies. Table 2.2 shows an example of poor accuracies presented by Jagannathan et al. [23]. The accuracy is very dependent on the chosen value of $\epsilon$. Random decision tree (RDT) classifiers provide the advantage that no privacy money is necessary for the tree growing process because it can be performed without any data query. Only the leaves need to be made privacy preserving. This is easily done because the node statistics can be viewed as queries over the training data [23]. Less queries imply less noise addition. As a result, RDTs have been shown to be more efficient and provide better security than ID3 induced trees in the context of differential privacy [41]. An alternative to RDTs is to build decision tree branches on the basis of "noisy maximal [class] votes" instead of other intensive impurity metrics like the entropy. In combination with an ensemble method with bootstrap samples and a certain "privacy budget allocation strategy", this leads to accurate and stable classification results [29].

2.5.4 Privacy Preserving Evaluation

The field of privacy-preserving decision tree evaluation is different, yet somewhat related to the already discussed distribution models. Here, a server has a sensitive model and a client has sensitive feature vectors as input. The goal is to classify the client's data while the sensitive inputs (model and query) remain hidden from the counterparty.

As one of the firsts, Brickell et al. have provided a private classification program explicitly, which takes a binary decision tree as input. The program is secure against semi-honest attacks and is more efficient than those obtained by direct application of generic SMC techniques. It combines cryptographic techniques such as additive homomorphic encryption (Pallier encryption [34] in particular) and Yao's garbled circuits method [49]. Each decision node is replaced by a small garbled circuit that is used to compare the client's attribute value and the decision node's threshold (both made unrecognizable for the other party), in a such way that the client only learns one of two keys, depending on the comparison. The revealed key includes the identifier of the next node in the evaluation path and its decryption key. The client can only evaluate one path, which appears random to the client. The program is structured in in three phases:
1. **Creation of the secure branching program:** In an offline procedure, the server builds a secure branching program, consisting of a garbled circuit for each inner node, which refers to a next node, and encrypts the class labels of each leaf node.

2. **Oblivious attribute selection:** In the second phase, the parties apply an additive homomorphic public-key encryption instance to supply the client with modified attribute values, which it can use as inputs to the compare circuits from the first phase.

3. **Evaluation of the secure branching program:** Finally, the client receives the secure branching program from the server and evaluates it locally.

Bost *et al.* [7] found out empirically that for comparison in the sense of Yao’s millionaire’s problem, specialized homomorphic cryptosystems are more efficient than garbled circuits, while garbled circuits are the more efficient option for comparison of unencrypted values. In their work [7], they use a fully homomorphic (additive and multiplicative) encryption based comparison protocol in a private decision tree classification algorithm. The algorithm is based on a polynomial representation of a tree, requires only a small number of multiplications and is based on "SIMD slots". SIMD slots denote a method that allows encrypting multiple bits in a single ciphertext and applying any operation to each of the bits in parallel by applying it to the ciphertext [40]. The tree is expressed as a high-degree polynomial that represents the sum of terms, where each term corresponds to a path in the tree from the root to a leaf node. The client encrypts its input and the server evaluates the encrypted input by exploiting the homomorphy properties. To face slow performance of the encryption scheme, the authors introduce a technique to reduce the length of the polynomial from the depth of the tree to its binary logarithm, which makes fully homomorphic evaluation significantly more efficient. The output of the polynomial is the classification result of a client.

In 2016, Wu *et al.* [22] published a protocol and claim that their protocol can evaluate a 1100-node tree in 30 seconds and requires about 10 MB of communication, representing a 10-fold improvement in computation and a 2.5-fold reduction of the bandwidth, compared to the protocol of Brickell *et al.* [9]. To avoid using heavy fully homomorphic encryption, the method of Wu *et al.* requires the server to transform the decision tree into a randomized one and send it to the client. This security restriction leads to an exponential complexity in relation to the depth of the tree. The protocol still uses additive homomorphic encryption.

Tai *et al.* [43] obtain a further speed improvement using another technique to replace fully homomorphic encryption by an only additive homomorphic one. Instead of representing a decision tree as a high-degree polynomial like Bost *et al.* [7], they represent it in the form of linear functions and therefore make the computation run-time proportional to the number of nodes. Both Wu *et al.* [22] and Tai *et al.* [43]
provide a protocol for both semi-honest and malicious parties and append notes to support random forests.
Client/Vertical-Server Set Intersection

The client/vertical-servers classification protocols in Section 4.2 and 4.3 require a special kind of private set intersection protocol as a central building block. They deal with a client/vertical-servers architecture, a novel scenario that is a hybrid of vertically distributed data and the client-server architecture of private evaluation in the context of classification (and regression). This chapter provides a client/vertical-servers set intersection protocol (CVSSIP) that is tailored to this architecture in order to offer an efficient tool without heavy-weighted cryptographic primitives. At first, the requirements for the CVSSIP are described more precisely. Afterwards, the algorithm is presented in detail followed by a closing privacy and complexity analysis.

3.1 Problem Statement

Assume $n$ servers $p_i$ own a private set and want to supply a client with their jointly set intersection but nothing else. The proviso is that each $p_i$ must not disclose any information about its input to one of the other servers. The private sets are represented by a binary vector $D_i \in \{0, 1\}^s$ for each $p_i$ with $D_i[j] = 1$ if $p_i$ supports the element $j$ and $D_i[j] = 0$ else. The output is a binary vector $Y \in \{0, 1\}^s$ with

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

(3.1)

Therefore, no vertical server $p_i$ is allowed to reveal any value of $D_k$ or $Y$, $k \neq i$. Let $T$ be a collusion threshold parameter. If less than $T$ servers collude with each other and $cl$ in addition, they must not be able to induce input values of any other server. $cl$ receives $Y$ but must not learn anything else.
This problem can be solved by calculating each $Y[j] \in Y$ independently from the others via a zero-sharing method. In zero-sharing, the servers distribute random seeded numbers that sum up to zero. Then, every $p_i$ sends its shares to $cl$ if $D_i[j] = 1$, else it sends a new random number. If the sum of all values is zero, $cl$ knows that $Y[i] = 1$. Since the distributor of the random shares could collude with $cl$ and they together would be able to retrace whether $D_i[j]$ of an individual party is one or zero, every party is a distributor of $T$ random shares.

Algorithm 4 explains this step 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_{i0}, \ldots, r_{iT}\}, where $T \in [1, n-1]$ (lines 3-5), sends its sum to $cl$ (lines 7-8) and
scatters \( \{r^1_i, \ldots, r^T_i\} \) to \( T \) other servers (line 6 and 10). Then, every \( p_i \) sends the sum of \( r^0_i[j] \) 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, and otherwise all of the servers hold \( d \) with a probability of \( 1 - 1/2^b \). The probability of a false positive is therefore negligible. For the client/vertical-servers classification purposes, the size of an intersection should be always one (see Chapter 4). Hence, the unlikely occurrence of a false positive can be detected easily and the procedure restarts for the candidate elements with new random seeds.

Table 3.1 gives an example of the zero-distribution and -composition process within the outer loop for the values \( n = 4, T = 2, \prod_{i=1}^{n} D_i[j] = 1 \) and arbitrary random generated values for every \( r^t_i[k] \). The columns determine the creating, calculating or sending party, the rows determine the receiving party. The values in brackets are only locally relevant and not communicated.

### 3.3 Security Analysis

For the purpose of analyzing the information leakage of Algorithm 4, questions of particular interest are:

- Which information can a semi-honest/malicious vertical server acquire?
- Which information can several vertical servers acquire?
- Which information can a semi-honest/malicious client acquire?
- Which information can a client acquire when it colludes with less/more than \( T \) vertical servers?
In the following, I will answer these questions separately by looking back at the communicated values in Algorithm 4 and Table 3.1:

- **Information gain of any number of colluding vertical servers:** The only messages a \( p_i \) receives are different zero-shares, \( r_{t-1}^i[j] \forall t, j \) (line 11). They are randomly generated independently from any input data. Hence, any number of colluding servers is not able to disclose private data, no matter whether they behave semi-honest or malicious.

- **Information gain of jointly colluding vertical servers and the client:** The situation changes if a client (\( cl \)) is involved in a collusion with some vertical servers. The accomplices might try to find out whether an element \( d \in D \) is supported by all parties or whether a particular vertical server \( p_x \) holds \( d \). The question if \( D_i[d] = 1 \) for all parties is defined by \( \sum_{i=0}^n (R_i - S_i) \). Since each \( S_i[j] \) is directly sent to \( cl \), only \( cl \) is able to learn it. In order to find out whether a particular vertical server \( p_x \) holds \( d \), adversaries have to know if:

\[
S_x = \sum_{t=0}^{T} r_{x-t}^t \mod n \Leftrightarrow S_x = \sum_{t=1}^{T} r_{x-t}^t \mod n + r_0^0. \quad (3.2)
\]

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 client. 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 \( p_x \). \( r_0 \) 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 \( cl \) and \( \sum_{t=1}^{T} r_x^t \) can only be calculated by the \( T \) servers on the left side of \( p_i \).

- **Information gain of the client:** From the former point follows that a \( cl \) cannot reveal any information about the inputs of the protocol without the support of other parties. It makes no difference whether \( cl \) is semi-honest or malicious because it cannot manipulate the messages it gets (lines 9 and 14) since each \( R_i[j] \) and \( S_i[j] \) is independent from any input of \( cl \).

In conclusion, at least \( \min(n-1, 2T) \) colluding vertical servers (the \( T \) on the "left" and on the "right" side of \( p_x \)) and the client are necessary to find out whether a particular party \( p_x \) supports an element \( d \). Furthermore, if \( cl \) represents the data record \( j \), it should not be of its interest to disclose this information. Otherwise, the users have to ensure the trustworthiness of \( cl \) or choose a sufficiently large \( T \) to prevent this attack. For example, the commodity server of a client/vertical-server classification framework functions as \( cl \) in this algorithm and collects data of multiple foreign records (see Section 4.2). Among other application-related privacy aspects, this is discussed in Section 4.4.
3.4 Complexity Analysis

In the first part of Algorithm 4 (lines 3-12), every server generates $T + 1$ random numbers, sends $T$ of them to other servers, sums them up, and sends one number to the client. Thus the computation and communication costs for one vector element and one server is in $O(T)$, the communication costs of the client is in $O(n)$. Afterwards (lines 13-14), each server either sums $T + 1$ numbers up or generates a random number. The result is sent to the client. Hence the communication costs are constant for each $p_i$ and in $O(n)$ for the client. The servers’ computation costs are in $O(T)$ again. In the last step (line13), the client makes $2n$ additions and one subtraction. In conclusion, both the communication and computation costs are in $O(s \times n \times T \times b)$, with the vector length $s$, the number of vertical servers $n$, the security threshold parameter $T \in [1, n]$ and the bit length of a number $b$. They compose of $O(s \times T \times b)$ elementary operations of each $p_i$ and $O(s \times n \times b)$ elementary operations of cl. Due to the independent calculation of each $Y[j]$ the algorithm is highly parallelizable.

Refering to Section 2.4, the two-party scalar product protocol of Du and Zahn [15] requires $O(s \times b)$ elementary computation operations, which is of the same order as Algorithm 4 with $n = 2$. The Paillier encryption version consumes $n(s + 1)$ multiplications for the encryption and summation and one modular exponentiation for the decryption. Note that the operations on encrypted data are done with values whose bit length corresponds to the bit length $B$ of the public-key and hence, are far slower that the other ones. The German federal office for information security recommends a key length of 2,000 to 3,000 bits for public-key cryptosystems [10]. A non-private simple straight-forward set intersection protocol requires $(n - 1) \times s \times b$ operations, because every entry of $Y$ is calculated by $n - 1$ multiplications. Table 3.2 shows the communication costs depending on the bit length $b$ of an uncoded value or $B$ respectively, assuming that the result is made available for a client.

<table>
<thead>
<tr>
<th>Intersection protocol</th>
<th>2 vertical servers</th>
<th>$n$ vertical servers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Du02</td>
<td>$4s \times b$</td>
<td>-</td>
</tr>
<tr>
<td>CVSSI</td>
<td>$6s \times b$</td>
<td>$s \times n \times (T + 2) \times b$</td>
</tr>
<tr>
<td>Paillier encryption</td>
<td>$2s \times B$</td>
<td>$s \times n \times B$</td>
</tr>
<tr>
<td>Non private</td>
<td>$2 \times s \times b$</td>
<td>$n \times s \times b$</td>
</tr>
</tbody>
</table>

Tab. 3.2: Communication costs
Random Forest Computation in a Client/Vertical-Servers Setting

This chapter is about the classification problem in the hybrid context of vertically partitioned data and the client-server architecture. First, the scenario of client/vertical-servers classification is described more precisely. Subsequently, I present a random forest training algorithm and a distributed random forest private evaluation algorithm, which solve the novel problem statement. Like conventional vertically partitioned data classification protocols [15, 45], these algorithms employ a secure set intersection protocol to preserve privacy in this context. However, the model properties of the scenario enable a simple solution that is fast working and scalable as well. This chapter is complemented by a security and complexity analysis.

4.1 Problem Statement

Problem description:  I consider the client/vertical-servers architecture as a composition 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 an existing private model. In the training phase, there are $m$ 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. The servers train a distributed model $f(x)$ based on the input vectors. In the evaluation phase, the query of a client $c$, whose input data $X[c]$ is distributed the same way as the training records, 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 $X_i[j]$ other than what can be deduced from $f(c)$. I allow the use of a commodity server $cs$. However, it does not strictly meet its conventional definition, because it does not act as as an initializer of a secure communication but rather functions as a central communication interface.
Fig. 4.1: Example of a random tree skeleton.

for all cooperating parties. Nevertheless, \( cs \) must not collude with any \( p_i \) and receives nothing but anonymous data.

4.2 Random Forest Training Protocol

Algorithm 5 presents the training steps of a private random forest. The model is distributed over all vertical servers and a commodity server. Each \( p_i \) receives the same tree skeletons (lines 1-3). It maps an identifier and a party \( p_i \) to each branch similar to the model already proposed by other authors [42, 45]. Figure 4.1 illustrates an exemplary tree skeleton. In the example, \( p_1 \) and \( p_2 \) know that \( p_i \) is responsible for the nodes 1, 3, 4 and 5 and \( p_2 \) for the others, but they do not know which attributes belong to the nodes and which attribute values are assigned to which edges. The leaves are only labeled with identifiers. Only the commodity server obtains a mapping from a leaf’s identifier \textit{leafID} to its class label/distribution in a posterior step. Every server maps selected attributes to each node randomly (lines 4-6), which corresponds to the assignment of a random forest whose parameter \( s \) (see Section 2.1) is set to one. Table 4.1 presents an example of the private labels of the tree in Figure 4.1. \( p_1 \) labeled the nodes 1,3,4 and 5 and \( p_2 \) the other ones. None of them knows which attributes the counterparty uses and how the edges are labeled that are incident to the counterparty’s nodes. 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 of the leaves. It receives an assignment of a leaf node to a class label for each instance and for each decision tree of the random
input: An attribute vector $X_i[j]$ for each training instance $j \in \{1, ..., m\}$ of each vertical server $p_i$, $i \in \{1, ..., n\}$, a collusion threshold parameter $T$, a commodity server $cs$, a server $p_c$, that holds the class values $C[j] \in C$, the number of trees in a forest $o$

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

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

Algorithm 5: Client/Vertical-Servers Random Forest Training

forest. As long as $cs$ does not collude with any vertical server, it cannot associate any feature or class label with the identity of any instance. Note that the $m$ training records are randomly drawn with replacement for each tree, since Algorithm 5 is a random forest algorithm and therefore includes the technique of bootstrap sampling. However, the training records can be chosen by an arbitrary $p_i$, which broadcasts the list of training instances to the other ones. Next, Algorithm 4 generates the leaf node that is provided to $cs$. The commodity server is treated as a client in Algorithm 4. First, each $p_i$ performs a preprocessing step: It assigns $D_i[l]$ with one if the instance $j$ reaches the leaf $l$ (based on the features in $X_i[j]$), or zero if not (line 10). Let $\{D_i\}$ denote the collection of private vectors $D_i$ of all $p_i$. $\{D_i\}$ and the collusion threshold parameter $T$ are provided as the input for Algorithm 4 (line 11). The output is the leaf index, $leafID$, of the record $j$. The commodity server updates the class distribution statistics of $leafID$ with $f(j)$ that it receives from the class server $p_c$ (line 12).
**Algorithm 6:** Client/Vertical-Servers Random Forest Classification

4.3 Random Forest Classification Protocol

To classify a test instance \( c \) of a client whose feature vector \( X[c] \) is distributed on the vertical servers, Algorithm 6 can be applied. Next to \( X[c] \), it requires a suitable classification model as input, consisting of a distributed tree ensemble and a commodity server with a mapping that maps the index of each leaf node to a class label and a decision tree ensemble. Algorithm 5 builds such a model. In the offline part (lines 2-4), all vertical servers \( p_i \) need to initialize a vector \( D_i \) with one at \( D_i[l] \) if \( c \) reaches the leaf \( l \) and with zero otherwise for all \( l \). Afterwards, Algorithm 4 is employed to find the unique leaf that all parties have in common in \( \{D_i\} \) (line 5). Client \( c \) conforms to \( cl \) in Algorithm 4, so it receives the leaf ID \( = c.leafIDs_{tree} \) corresponding to its features for each decision tree. Subsequently, it sends a request with all the leaf IDs to the commodity server to receive the most likely class label (line 7). This is simply done by returning the class value that is mapped to all leafIDs most frequently or, in case of a regression, the weighted average of all mapped values.

**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 \( c \). Subsection 4.6.2, describes an alternative method to preserve the client’s privacy, which can be applied when practical obstacles impede an anonymous communication among a client and the commodity server. Moreover, \( c \) should get a shared one-time-password by one or more parties to prevent it from sending multiple malicious requests with different leaf IDs to \( cs \), such that it is not able to deduce sensitive information about
the model and the underlying data. This procedure can be realized by a server supplying a sufficient number of clients with an one-time password each, permuting the list of passwords and forwarding them to cs. A client can make a request with a valid password out of this list, but cs cannot associate them with individual clients. If all or a sufficiently large number of servers agree on the passwords and cs validates that all servers sent the same passwords, it is ensured that there exist only as much passwords as clients exist. This limits the risk that a client and a $p_i$ could collude and send multiple requests.

4.4 Security Analysis

This section gives an analysis of the robustness of Algorithm 5 and 6 to information leakage. In Algorithm 5 (line 16) and Algorithm 6 (line 5), the interactions among servers are limited to the interactions in Algorithm 4, hence, the security aspects of Algorithm 4 are directly transferable to them. Here, only some high-level aspects that complement Section 3.3 are considered:

- **Association of training records with their leave nodes (inference problem):** In case of a collusion among cs and a vertical server, the collaborators can associate the identities of all the training records to their corresponding sensitive leaf nodes and therefore, similarities between the records and their strongly related class labels as well. That is the reason for the requirement of having a trustworthy commodity server that does not collude with any vertical server. Despite this restriction, using a commodity server improves the run-time effectively, and – according to Du and Zahn [15] – finding such a cs is feasible in practice. Before supplying cs with the sensitive class labels, the class server could hash them in order achieve additional security against a collusion among a commodity server and any other server. But with some background information and an analysis about the frequencies of the individual labels, the critical values can be inferred anyway.

- **Disclosure of the feature values of other vertical servers or the leaves belonging to a particular instance:** In the multi-party setting, there is a general risk of collusion among 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, assume that $u$ colluding vertical servers try to reveal the input data of one or more other servers or the leaves belonging to a particular instance. Since the interactions among any servers in Algorithm 5 and 6 are limited by Algorithm 4, the security aspects about input and output of the CVSSIP apply: The output is only accessible for cs in
the training mode and c in the classification mode respectively, and in order to infer an other server’s input, a semi-honest or malicious collusion of both at least T servers and cs or respectively c is necessary.

- Disclosure of the identity of a client to the commodity server: In line 7 of Algorithm 6, c receives the final classification output from cs. Assuming that this communication is maintained anonymous, cs can not associate c with the final leaf node and class label, which it passes uncoded. Section 4.6.2 handles the situation if this assumption cannot be made.

- Disclosure of the model to a client: A client might try to send multiple maliciously modified requests to cs in order to deduce information about the model and the underlying data. A countermeasure against this attack is described in Section 4.3.

### 4.5 Complexity Analysis

Algorithm 5 starts with the building of k trees with $\beta^\delta$ nodes (line 1-5), where $\beta$ is the constant number of splits per node and $\delta$ the depth of a tree. Section 4.6.1 explains the advantages of a fixed $\beta$, but still describes how this restriction can be liquidated. Each party receives the nodes for which it is responsible. In total, $\beta^\delta$ node IDs minus the number of nodes of the distributing party itself have to be communicated. The next steps (lines 6-9 and 11-15) proceed completely offline. In total, the servers label $k \times \beta^\delta$ nodes and determine for $m$ records which leaves can be reached with respect to their individual knowledge. Hence, $m \times \beta^\delta$ comparing and storing operations per party is a rough upper bound, but the average number is far lower in practice. In summary, the computation costs of this preprocessing steps are in $O(m \times n \times \beta^\delta)$ operations and the communication costs are in $O(\beta^\delta)$ elementary data objects.

In the online part (line 16), Algorithm 4 is called for $m \times o$ times with the input size $\beta^\delta$, which results to a computation complexity of $O(m \times n \times o \times T \times \beta^\delta \times b)$ bit operations and $m \times o \times n \times (T + 2) \times \beta^\delta \times b$ bits to communicate. Finally, the class server sends the class label of the $m$ records to the commodity server. The costs of Algorithm 6 are quite similar. The step of building and labeling a forest is omitted. But the determination of all candidate leaves and the call of Algorithm 4 accord with $m = 1$. The overall complexity of a straight-forward non-private version, where all locally supported records are broadcast to every vertical server at each tree node once, is $O(m \times n \times o \times \beta^\delta)$, which is smaller than the version with the CVSSIP by the customizable factor $T$ only. But it applies especially to the non-private version that the number of supported instances in the deeper nodes is far smaller than $m$. 

32 Chapter 4 Random Forest Computation in a Client/Vertical-Servers Setting
4.6 Extension Notes

In this section, I discuss three implementation-dependent aspects that can lead to extensions of the client/vertical-servers architecture: the number of partitions that are made in a branching step in Algorithm 5, the communication among the client and the commodity server in Algorithm 6, and the meta algorithm boosting to decrease the communication and computation costs of Algorithm 6 especially.

4.6.1 Number of Splits per Branch

In Algorithm 5, the random trees are created with a constant number of splits ($\beta$) in every branch node. A fixed split size provides some benefits:

- The party that generates the random forest skeleton (line 3 of Algorithm 5) does not need to know all split sizes to consider them in the building process. Furthermore, the consideration of different attribute sizes in the first step would determine which attributes have to be used by the individual parties and reduce their possibilities to consider background knowledge in the attribute assignment step.
- The split size can indicate the belonging attribute to an adversary with background knowledge. A homogeneous size prevents this.
- Broader splits lead to record partitions of smaller sizes than narrower ones, which influences their weights in a certain way. Attributes with an equal number of values have consistent weights.
- A fixed $\beta$ leads to trees with a fixed number of $\beta^d$ leaf nodes and, hence, a fixed input size for Algorithm 4.

However, a constant $\beta$ is not naturally given, because different attributes can have different specificities. Usually, the number of splits in a branch corresponds to the number of discrete values that an attribute has. This applies especially for nominal attributes. For reasons of simplicity and effective information gains, real numbered values are often split into two binary at a node, but can be split again with a new threshold at a deeper node. This procedure is also adaptable to discrete valued attributes: If possible, the feature values should be sorted and transformed to numeric values; otherwise it has to be differentiated between attributes with less or more than $\beta$ values. Attributes with less values can be complemented with dummy values, so that they are able to split records into $\beta$ partitions even if the dummy partitions will be empty. The values of attributes with more than $\beta$ values have to be put in $\beta$ buckets. There is no generally best solution to design this step. One practical way is to use random assignments here as an additional element of variance in the random forest.
Despite all drawbacks, Algorithm 5 can be modified to support dynamic branch sizes. Therefore, the servers label their attributes with an identifier and send the identifiers, linked with their number of accepted values, to the server that builds the random forest skeleton. This server needs to label all nodes with the attribute classifier in the skeleton building step already, such that the different branch sizes can be considered. In the following step (line 8), the servers only replace the identifiers by the corresponding features instead of performing an own random attribute assignment.

### 4.6.2 Classification with Oblivious Transfer

In line 7 of Algorithm 4, the client \((c)\) sends a request including its leaf IDs to the commodity server \((cs)\) and receives its classification result from \(cs\) in return. The protocol discloses no background information about \(c\) to \(cs\), but the communication network or other case specific factors could reveal the clients’ identity. In such cases that impede an anonymous communication among a client and the commodity server, this communication part can be done via a string-select oblivious transfer protocol, such that the commodity server does not learn the input of the client (leafIDs) and the output of the protocol.

In cryptography, an oblivious transfer (OT) is a type of protocol in which a sender transfers one out of several pieces of information to a receiver, but remains oblivious which piece has been transferred. The first OT developments [37, 17], are based on the public-key encryption scheme RSA. Even, Goldreich and Lempel [17] provided a 1-out-of-2 oblivious transfer in order to use them in SMC protocols. In addition to the general requirement above, in a 1-out-of-2 OT, the information \(m_b\), the receiver obtains depends on a bit \(b\), which it has sent to the sender before. The 1-out-of-2 OT must ensure that the sender does not learn \(b\). In the RSA based solution, the procedure consists of a few more rounds, so that the receiver sends \(b\), which has been modified by random values of both parties respectively. At the end of the protocol, the sender sends two values to the receiver of which it can only decrypt \(m_b\). In an 1-out-of-n OT protocol, the receiver sends a value between 1 and \(n\) instead of \(b\) and decrypts the message with the index of this value. Kolesnikov et al. [26] provide an efficient 1-of-n oblivious transfer protocol, which requires only four times the costs of an 1-out-of-2 oblivious transfer in an amortized setting and therefore is highly scalable.

There are several options to apply the efficient 1-out-of-n OT to Algorithm 4 with different pros and cons: First, \(c\) can concatenate the leaf IDs of all trees to one long bit-string and input it into a single OT. This would require that \(cs\) has already saved the final outputs for all possible bit-strings, which is not feasible in practice. Secondly, \(c\) and \(cs\) can run one OT for each of the \(o\) trees of the random forest, such that \(c\) receives \(o\) class distributions and can combine them to a final result self-sufficiently.
The disadvantage is that the communication costs increase by the factor $o$ and $c$ gets more information about the classification mode, because it receives its own output for each single tree.

### 4.6.3 Boosting

Random forest algorithms benefit from a high robustness against noise and overfitting and a fast training procedure. However, in order to achieve a good performance, they require a high number of trees, because many of them could be more or less redundant. The high number of trees causes a high memory consumption. In Algorithm 6, the communication costs are proportional to the number of trees. Hence, it would be desirable to reduce the number of necessary decision trees. Mishina, Tsuchiya and Fujiyoshi [33] propose a boosted random forest that reduced the amount of used memory by 47% compared to a conventional random forest in their experiments while they maintained the performance. Boosting [38] is a typical method to combine weak learners in order to construct a more accurate classifier than the individual ones. It is a sequential training of classifiers in which the classification errors of a learner are used to overweight previously wrong classified records of the training set of the subsequent learner. The training instances are reweighted for a fixed number of times. Additionally, the individual classifiers are weighted, usually by their accuracy. In a strict sense, only provable probably approximately correct learning [47] formulations belong to the boosting algorithms [32].

Since the random-forest parameter $s$ (the number of randomly drawn attributes in each branching step) is always one in Algorithm 5, which tends to trees with more insignificant attributes and can reduce the classifiers performance slightly (see Subsection 5.2), the positive effect of adjusting the weights of different trees might be particularly interesting for Algorithm 5, because weaker classifiers become underweighted. The boosted random forest approach is integrable in Algorithm 5 as follows: Algorithm 5 is executed with $o = 1$ to build a weak classifier with one decision tree. Then, the class server acts as a client for all training instances and receives their predictions. Since it knows the ground truth of all records, it can calculate the misclassification error and update the tree weights and reweight the records. This is repeated several times for more trees. Subsequently, the class server sends all tree weights to the commodity server. In the classification phase, the commodity server returns the trees’ weighted average class distribution instead of the unweighted one.
This chapter evaluates the feasibility and scalability of the client/vertical-servers algorithms that are presented in this thesis. The first section describes the test conditions and parameters including hardware properties, benchmark algorithms, variable setting and data sets. Subsequently, the run-time of Algorithm 4 as a stand-alone building block is evaluated separately, and then the run-time measurements of Algorithm 5 and 6 for different real world data sets are presented. Moreover, the impact of the complete renunciation of an impurity function in the random forest is tested and how the run-times of Algorithm 5 and 6 spread on different program parts.

5.1 Experiment Settings and Datasets

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

1. The CVSSIP as designed in Algorithm 4.
2. The version Du02 where I used a modified version of the scalar product protocol by Du and Zhan [15], such that the commodity server receives the output. This version has the constraint of a commodity server like the CVSSIP 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 context of decision trees over distributed data [45, 11], because it fulfills the requirement of additive homomorphy. In the modification, a first party encrypts the index of \( d \) if it supports the element \( d \) else zero with the public-key. Then, each vertical server multiplies the encrypted intermediate result of its predecessor with one if it supports \( d \) and with zero if it does not. Both multiplications rerandomize the encryption such that the multiplied factor is unrecognizable for every other party. At the end, the results for each \( d \) are summed up, such that the total result is the encryption of the index of \( d \), because in the application case of this thesis, only one \( d \) (the unique leaf ID) is supported by all parties together. Only the commodity server has the private key and can decrypt the result. For simplicity,
I did not use the state-of-the-art public-key encryption, but give an idea of homomorphic encryption techniques.

4. A procedure with public splits like by Suthampan and Maneewongvatana [42] instead of a private set intersection method as a baseline. The method is very straightforward, but reveals information the other versions intend to protect.

All experiments were executed on a single device with a dual core intel i7-5500U cpu and a 8GB RAM. For the current results, I did not use a framework to simulate bandwidth and latency of a network of different devices. I tested the scalability on different real-world datasets of the UCI Machine Learning Repository [12] with different parameters: the number of vertical servers $n$, the number of trees in a forest $o$, the collusion threshold parameter $T$, and the number of leaf nodes. The number of leaf nodes is $\beta^\delta$, because the tree depth $\delta$ and the number of splits in a branch $\beta$ are fixed in the experiments. In Subsection 5.2.1, all times (except those in Figure 5.1) are averaged over 100 repetitions, in Subsection 5.2.2, all experiments are executed within a stratified ten-fold cross validation and the times are given for an average fold (unless otherwise stated).

5.2 Experimental Results

5.2.1 Client/Vertical-Servers Set Intersection

Table 5.1 compares the three set intersections versions CVSSI, Du02 and Pallier for different vector lengths $s$ detached from the deployment application. It verifies the theoretical complexity level of $O(s)$ of both the Du02 and Pallier version in a two-party setting. It also shows that the Du02 version performs up to six times faster.
than the CVSSIP, which might be due to more effective vector operations in my implementations. Further implementation improvements might be able reduce the constant factor. The Paillier version scales proportional to $s$ too, but it is generally far slower because the algorithmic operations are in a field of the size $B$, where $B$ is simultaneously the key bit length and needs to be sufficiently large. In Table 5.1 $B$ is set to 2,000, the concurrent recommendation of the BSI [10].

Figure 5.1 studies the run-time behavior of the Pallier set intersection algorithm as a representative for public-key encryption schemes and indicates a polynomial dependency on $B$. The run-time for a single small vector with 1,000 elements is a few minutes. Vaidya and Clifton came to similar results [45]. This is rather unfeasible for the whole tree building and classification procedure. The CVSSIP requires 1 ms for this task. The recently published private set intersection protocol by Kolesnikov et al. [26] 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 used a much more elaborate framework to simulate communication costs than I did.

Next, the scalability of the CVSSIP is inspected more specifically. The Figures 5.2, 5.3 and 5.4 vary the vector length, number of parties and the collusion threshold respectively and confirm that the run-time is proportional to $n$, $o$ and $T$. They also show that the vectors of eleven parties with a million entries each can be processed in a few seconds, which demonstrates that the protocol is feasible for larger vector sizes and a higher number of involved servers.
Fig. 5.2: Impact of the vector length $s$ on the run-time of the CVSSIP, with $n = 4$ and $T = 3$

Fig. 5.3: Impact of the number of vertical servers $n$ on the run-time of the CVSSIP, with $s = 10^6$ and $T = 1$

Fig. 5.4: Impact of the collusion threshold parameter $T$ on the run-time of the CVSSIP, with $s = 10^6$ and $n = 11$
### 5.2.2 Client/Vertical-Servers Random Forest

This subsection summarizes the test results of a whole training and testing scenario on different real world data sets, partitioned in training (90% of the records) and private evaluation sets (10% of the records). I successfully validated the output (accuracy and confusion matrix) of the scenario by comparing it with the output of a non-private random forest implementation of the WEKA workbench [20] in some sample checks, to make sure that the SMC requirement of equally results is fulfilled. A little constraint is that referred to a classical random forest the parameter $s$ is always one. Table 5.2 shows the results of a small experiment on how the choice of $s$ can influence the accuracy of a random forest algorithm. It indicates that higher values of $s$ than $s = 1$ can lead to a slightly higher percentage of correctly classified examples (nursery data set e.g.), but this can not be seen as a general rule. Compared to randomization techniques the reduction of the classification performance is marginal.

<table>
<thead>
<tr>
<th>dataset name</th>
<th># features $s$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>cars</td>
<td>93.52</td>
<td>94.61</td>
<td>94.79</td>
<td>94.73</td>
<td>94.38</td>
<td>94.56</td>
<td></td>
</tr>
<tr>
<td>contraceptive</td>
<td>50.71</td>
<td>51.26</td>
<td>52.07</td>
<td>52.20</td>
<td>51.26</td>
<td>51.93</td>
<td></td>
</tr>
<tr>
<td>nursery</td>
<td>98.40</td>
<td>98.90</td>
<td>99.00</td>
<td>99.03</td>
<td>99.06</td>
<td>99.07</td>
<td></td>
</tr>
<tr>
<td>phishing websites</td>
<td>88.29</td>
<td>88.35</td>
<td>88.44</td>
<td>88.36</td>
<td>88.36</td>
<td>88.20</td>
<td></td>
</tr>
<tr>
<td>thoracic surgery</td>
<td>71.06</td>
<td>71.49</td>
<td>71.91</td>
<td>71.06</td>
<td>70.64</td>
<td>70.85</td>
<td></td>
</tr>
</tbody>
</table>

Tab. 5.2: Percentage of correctly classified test instances with different values of the random forest randomization parameter $s$, with $o = 100$

<table>
<thead>
<tr>
<th>number of trees</th>
<th>Paillier</th>
<th>CVSSIP</th>
<th>Du02</th>
<th>Without PSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>115.519</td>
<td>0.464</td>
<td>0.080</td>
<td>0.003</td>
</tr>
<tr>
<td>2</td>
<td>238.991</td>
<td>0.922</td>
<td>0.169</td>
<td>0.006</td>
</tr>
<tr>
<td>4</td>
<td>461.899</td>
<td>1.796</td>
<td>0.321</td>
<td>0.008</td>
</tr>
<tr>
<td>6</td>
<td>693.079</td>
<td>2.701</td>
<td>0.466</td>
<td>0.011</td>
</tr>
<tr>
<td>8</td>
<td>919.099</td>
<td>3.591</td>
<td>0.639</td>
<td>0.014</td>
</tr>
<tr>
<td>10</td>
<td>1,165.374</td>
<td>4.631</td>
<td>0.833</td>
<td>0.015</td>
</tr>
</tbody>
</table>

Tab. 5.3: Run-times of the client/vertical-servers random forest algorithms on the car dataset in seconds, with $n = 2, \beta^d = 4^3$ and $o = 5$
The main subject of this experimental analysis is the execution time of data sets with different properties:

- Table 5.3 contrasts the run-times of the four versions that are introduced in the beginning of this chapter on the small car dataset (1728 instances) with a small number of trees ($o = 5$) and a small number of leaves ($\beta^\delta = 1,000,000$) respectively in a two vertical server setting. As expected, the Paillier encryption version requires several minutes, although I set $B$ to the unacceptably low value of 64. Again, Figure 5.1 gives an idea, how this would change if $B$ is set to an acceptable public-key length. The approach of this thesis, CVSSIP, takes less than half a second per tree. Like in the unit test in Table 5.1, the Du02 version performs almost six times faster than Algorithm 4 but is still not applicable to n-party problems. The variant without private set intersection is very fast and suggests further potential for improvement, but suffers from the inference problem.

- Figure 5.5 visualizes the dependency of the run-time on the number of vertical servers in the example of the phishing websites set with 11,055 records. It confirms that the scaling behavior of Algorithm 4 (as investigated in Subsection 5.2.1) is deducible to the Algorithms 5 and 6 on a real world data set.

- Table 5.4 shows the run-time (in seconds) of CVSSIP on different real-world data sets with five parties, $T = 4$ and 20 randomly generated trees. The results, which vary from 16 seconds for a few binary splits and a small dataset to six and a half hours for the largest dataset with five splits per branch, suggest that the approach is feasible in practice.
In the final experiment, I studied how the total execution time of the client/vertical-servers architecture spreads on the different program parts. In order to make make Algorithm 5 and Algorithm 6 easier to compare, I divided the dataset (thoracic surgery), in a training and an evaluation set of the same size. Table 5.5 presents the computation times, in seconds averaged over ten repetitions, and Figure 5.6 gives a visualization of them. The program parts of Algorithm 5 for training (tr) and Algorithm 6 for classification (cl) are arranged in:

- Growing and broadcasting of the empty random tree skeletons (skel. (tr), line 3 of Algorithm 5),
- The determination of the candidate leaves of all parties (calc. $D_i$, line 14 of Algorithm 5 and line 3 of Algorithm 6),
- The use of the CVSSIP (line 16 of Algorithm 5 and lines 5 of Algorithm 6), and
- Administrative overhead (others).

<table>
<thead>
<tr>
<th>dataset name</th>
<th>$n$</th>
<th>$\beta$</th>
<th>runtime (in s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cars</td>
<td>1,728</td>
<td>$4^6$</td>
<td>155.7</td>
</tr>
<tr>
<td>contraceptive</td>
<td>1,473</td>
<td>$2^9$</td>
<td>16.8</td>
</tr>
<tr>
<td>hepatitis (without missing values)</td>
<td>80</td>
<td>$2^{17}$</td>
<td>11,507.4</td>
</tr>
<tr>
<td>nursery</td>
<td>12,960</td>
<td>$5^7$</td>
<td>22,783.6</td>
</tr>
<tr>
<td>phishing websites</td>
<td>11,055</td>
<td>$2^{13}$</td>
<td>16,231.7</td>
</tr>
<tr>
<td>thoracic surgery</td>
<td>470</td>
<td>$2^{13}$</td>
<td>173.1</td>
</tr>
</tbody>
</table>

Tab. 5.4: Run-times of the client/vertical-servers random forest algorithms on some UCI ML data sets with $n = 5$, $T = 4$ and $\omega = 20$

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>train. total</th>
<th>skel.</th>
<th>calc. $D_i$</th>
<th>CVSSIP</th>
<th>other</th>
<th>total (cl)</th>
<th>calc. $D_i$</th>
<th>CVSSIP</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>310</td>
<td>2</td>
<td>40</td>
<td>266</td>
<td>2</td>
<td>303</td>
<td>47</td>
<td>254</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>614</td>
<td>2</td>
<td>90</td>
<td>519</td>
<td>2</td>
<td>594</td>
<td>88</td>
<td>503</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1,201</td>
<td>2</td>
<td>174</td>
<td>1,021</td>
<td>3</td>
<td>1,180</td>
<td>175</td>
<td>1,004</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2,468</td>
<td>4</td>
<td>371</td>
<td>2,089</td>
<td>4</td>
<td>2,452</td>
<td>376</td>
<td>2,073</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>4,873</td>
<td>12</td>
<td>705</td>
<td>4,152</td>
<td>4</td>
<td>4,890</td>
<td>691</td>
<td>4,195</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>9,814</td>
<td>25</td>
<td>1,332</td>
<td>8,452</td>
<td>5</td>
<td>9,899</td>
<td>1,378</td>
<td>8,516</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>21,726</td>
<td>104</td>
<td>2,967</td>
<td>18,647</td>
<td>7</td>
<td>21,182</td>
<td>2,781</td>
<td>18,395</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>43,873</td>
<td>72</td>
<td>5,412</td>
<td>38,382</td>
<td>7</td>
<td>42,414</td>
<td>5,057</td>
<td>37,350</td>
<td>6</td>
</tr>
<tr>
<td>13</td>
<td>77,141</td>
<td>132</td>
<td>7,177</td>
<td>69,824</td>
<td>8</td>
<td>77,700</td>
<td>7,412</td>
<td>70,283</td>
<td>6</td>
</tr>
<tr>
<td>14</td>
<td>155,044</td>
<td>597</td>
<td>12,995</td>
<td>141,445</td>
<td>7</td>
<td>151,628</td>
<td>12,452</td>
<td>139,170</td>
<td>6</td>
</tr>
</tbody>
</table>

Tab. 5.5: Distribution of the computation time of the client/vertical-servers training (tr) and classification (cl) algorithms on the thoracic surgery dataset in seconds, with $n = 4$, $T = 3$, $\beta = 2^{11}$ and 50 trees
Figure 5.7 visualizes the relative shares of these components, based on the data of Table 5.5. As expected, the times of both algorithms are quite similar, since both employ the CVSSIP in the same way, and the proportion of the tree skeleton building process is negligible and decreasing because it is independent from the data sets. Figure 5.7 demonstrates that the slightly increasing proportion of the CVSSIP of the total time maintains relatively stable between 85% and 92% and builds the major part.

Fig. 5.6: Distribution of the computation time of the client/vertical-servers training (tr) and classification (cl) algorithms on the thoracic surgery dataset in seconds, with $n = 4, T = 3, \beta = 2^{11}$ and 50 trees

Fig. 5.7: Proportions of the computation time of the client/vertical-servers training (tr) and classification (cl) program parts on the thoracic surgery dataset, with $n = 4, T = 3, \beta = 2^{11}$ and 50 trees
Conclusion and Future Work

This thesis presents a novel problem formulation and architecture that is a hybrid 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 other organizations. This thesis provides a closed random forest architecture that solves the problem with a combination of different approaches used for related problems: private set intersection, zero-sharing, commodity server, private evaluation, etc. The random forest algorithm is robust against noise and enables a tree building without any data queries in the first step. Only the step of labeling the leaf nodes requires data base accesses.

To preserve privacy along the processing of a distributed data record, both the training and the evaluation algorithm employ the client/vertical-servers set intersection protocol (CVSSIP). It is a lightweight zero-sharing based algorithm that I designed as a tailor-made solution for the purpose of the client/vertical-servers classification and builds the core of this thesis. Nevertheless, private set intersection protocols are widely used in SMC and the CVSSIP is self-sufficient applicable. The CVSSIP is secure against joint attacks of $T-1$ malicious vertical servers and a malicious client on the input and output of other parties, whereby $T$ is a customizable parameter.

A central element in the random forest architecture is a commodity server, which functions as a communication interface. It collects the class label statistics of all leaf nodes in the training mode and returns the individual classification of a client's record with nothing but its leaf node identifier as the input in the evaluation mode. This thesis includes approaches on how to prevent clients from sending multiple malicious requests to the commodity server and on how to use a constant time 1-out-of-n oblivious transfer [26] in order to make the communication between client and communication server oblivious if necessary.

The architecture that I present in this thesis 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 [45],
2. Linear scaling with respect to the number of parties, the size of the data base, the number of trees, the number of leaf nodes per tree and $T$,
3. Accurate results in contrast to randomization based techniques [23, 18],

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, the framework of this thesis can handle distributed decision trees.

The experiments confirm that the solution is feasible in practice. E.g., it is able to train and use a model on a data set with 11,055 records, distributed to five parties, with 20 trees with more than 8,000 leaves each on a standard laptop in less than five hours. The test lacks a realistic network simulation on the one hand, but a suitable hardware that is able to exploit the high parallelizability of the algorithms might be able to gain faster results on the other hand.

In a classic random forest algorithm, the default value of the number of randomly chosen candidate features for a new branch node \( s \) is \( \lfloor \sqrt{|A|} \rfloor \) for classification [21], if \( |A| \) is the number of features. A restriction of Algorithm 5 is that \( s = 1 \) is fixed, which can reduce the performance slightly. Future work could inspect the impact in more detail and provide alternatives. In this thesis, I suggest to modify the random forest to a boosted random forest [33] to face the higher probability of insignificant features and to overweight stronger classifiers. The main drawback of this architecture is the assumption of a central non-colluding commodity server. Future work should address this. Making use of the results of Kolesnikov et al. on 1-out-of-n oblivious transfer and private set intersection [25, 26], may be a possibility to overcome this dependency in the future. Furthermore, research on the applicability of well performing non-branching classification algorithms to the client/vertical-servers scenario should be done.
Bibliography and Websites


List of Figures

2.1 Example of the variable values in Du's and Zahn's two-party scalar product protocol .................................................. 12
2.2 Data partitioning models: (a) vertically partitioned data, (b) horizontally partitioned data; (c) privacy preserving evaluation model .... 14
4.1 Example of a random tree skeleton.................................................. 28
5.1 Impact of the key bit length $B$ on the run-time of Paillier set intersection, with $n = 3$ and $s = 1,000$ .................................................. 39
5.2 Impact of the vector length $s$ on the run-time of the CVSSIP, with $n = 4$ and $T = 3$ .................................................. 40
5.3 Impact of the number of vertical servers $n$ on the run-time of the CVSSIP, with $s = 10^6$ and $T = 1$ .................................................. 40
5.4 Impact of the collusion threshold parameter $T$ on the run-time of the CVSSIP, with $s = 10^6$ and $n = 11$ .................................. 40
5.5 Impact of $n$ on the run-time of the client/vertical-servers random forest algorithms on the phishing dataset in minutes, with $n = 7, T = 6, \beta^\delta = 2^{11}$ and $o = 30$ .................................................. 42
5.6 Distribution of the computation time of the client/vertical-servers training (tr) and classification (cl) algorithms on the thoracic surgery dataset in seconds, with $n = 4, T = 3, \beta = 2^{11}$ and 50 trees .................................................. 44
5.7 Proportions of the computation time of the client/vertical-servers training (tr) and classification (cl) program parts on the thoracic surgery dataset, with $n = 4, T = 3, \beta = 2^{11}$ and 50 trees .................................................. 44
List of Tables

2.1 Impurity metrics for binary classes .............................................. 7
2.2 Example of poor accuracy results of \( \epsilon \)-diff. private ID3 [23] ........ 18
3.1 Example of variable values in a CVSSIP ........................................... 23
3.2 Communication costs ................................................................. 25
4.1 Example of two private trees belonging to the same tree skeleton ........ 28
5.1 Run-time of the two-party dot product and set intersection programs in seconds ............................................................ 38
5.2 Percentage of correctly classified test instances with different values of the random forest randomization parameter \( s \), with \( o = 100 \) ........ 41
5.3 Run-times of the client/vertical-servers random forest algorithms on the car dataset in seconds, with \( n = 2, \beta^5 = 4^5 \) and \( o = 5 \) ........ 41
5.4 Run-times of the client/vertical-servers random forest algorithms on some UCI ML data sets with \( n = 5, T = 4 \) and \( o = 20 \) ........ 43
5.5 Distribution of the computation time of the client/vertical-servers training (tr) and classification (cl) algorithms on the thoracic surgery dataset in seconds, with \( n = 4, T = 3, \beta = 2^{11} \) and 50 trees ........ 43
Declaration

I hereby declare that I have written the present thesis independently and without use of other than the indicated means. I also declare that to the best of my knowledge all passages taken from published and unpublished sources have been referenced. The paper has not been submitted for evaluation to any other examining authority nor has it been published in any form whatsoever.

Mainz, August 9th, 2018

_________________________
Derian Huibert Boer